GR5405开发者指南

版本： **1.1**

发布日期： **2025-04-22**

深圳市汇顶科技股份有限公司

商标声明

GOODiX 和其他汇顶商标均为深圳市汇顶科技股份有限公司的商标。本文档提及的其他所有商标或注册商标，由各自的所有人持有。

免责声明

本文档中所述的器件应用信息及其他类似内容仅为您提供便利，它们可能由更新之信息所替代。确保应用符合技术规范，是您自身应负的责任。

深圳市汇顶科技股份有限公司（以下简称“GOODIX”）对这些信息不作任何明示或暗示、书面或口头、法定或其他形式的声明或担保，包括但不限于针对其使用情况、质量、性能、适销性或特定用途的适用性的声明或担保。GOODIX对因这些信息及使用这些信息而引起的后果不承担任何责任。

未经GOODIX书面批准，不得将GOODIX的产品用作生命维持系统中的关键组件。在GOODIX知识产权保护下，不得暗中或以其他方式转让任何许可证。

深圳市汇顶科技股份有限公司

总部地址：深圳市福田区梅康路1号汇顶科技总部大厦26楼

电话：+86-755-33338828　　邮编：518000

网址：www.goodix.com

前言

编写目的

本文档主要介绍Goodix GR5405车载低功耗蓝牙系统级芯片（SoC）的软件开发工具套件（SDK），以及使用Keil/IAR开发和调试程序的方法，以帮助开发者开发车载应用。

读者对象

本文适用于以下读者：

- 芯片用户
- 开发人员
- 测试人员
- 技术支持工程师

版本说明

本手册为第2次发布，对应的产品系列为GR5405。

修订记录

版本	日期	修订内容
1.0	2024-08-30	首次发布
1.1	2025-04-22	• 增加Mesh相关描述。 • 更新“配套工具”章节。 • 更新“配置custom_config.h”章节。 • 更新bsp_log_init()示例代码和UART输出调试Log说明。

目录

1 简介

GR5405是Goodix推出的一款支持Bluetooth 5.3的车载低功耗蓝牙（Bluetooth LE）系统级芯片，能够在－40°C ～ 105°C的宽温度范围内稳定运行，通过 AEC-Q100 Grade2认证，可应用于数字钥匙、胎压监测等车载应用，并支持蓝牙Mesh网络协议。

基于64 MHz Arm® Cortex®-M4F内核，GR5405集成了2.4 GHz 射频（RF）收发机、低功耗蓝牙5.3协议栈、512 KB片内Flash、96 KB系统SRAM，以及丰富的外设。该芯片具有出色的RF性能，其最大发射（TX）功率可达+15 dBm，Bluetooth LE 1 Mbps模式下的接收（RX）灵敏度可达-99 dBm，整体链路预算（Link Budget）高达114 dB。

GR5405芯片还支持多从多主，可配置为广播者（Broadcaster）、观察者（Observer）、外围设备（Peripheral）和中央设备（Central），以及各种角色的组合应用。

1.1 GR5405 SDK

GR5405软件开发套件（Software Development Kit，SDK）为GR5405系列SoC提供全面的软件开发支持。该SDK包含Bluetooth LE API、Mesh API及System API等、外设驱动程序、调试/下载工具、工程示例代码以及相关的用户文档等。

1.2 低功耗蓝牙协议栈

低功耗蓝牙协议栈的架构如下图所示：

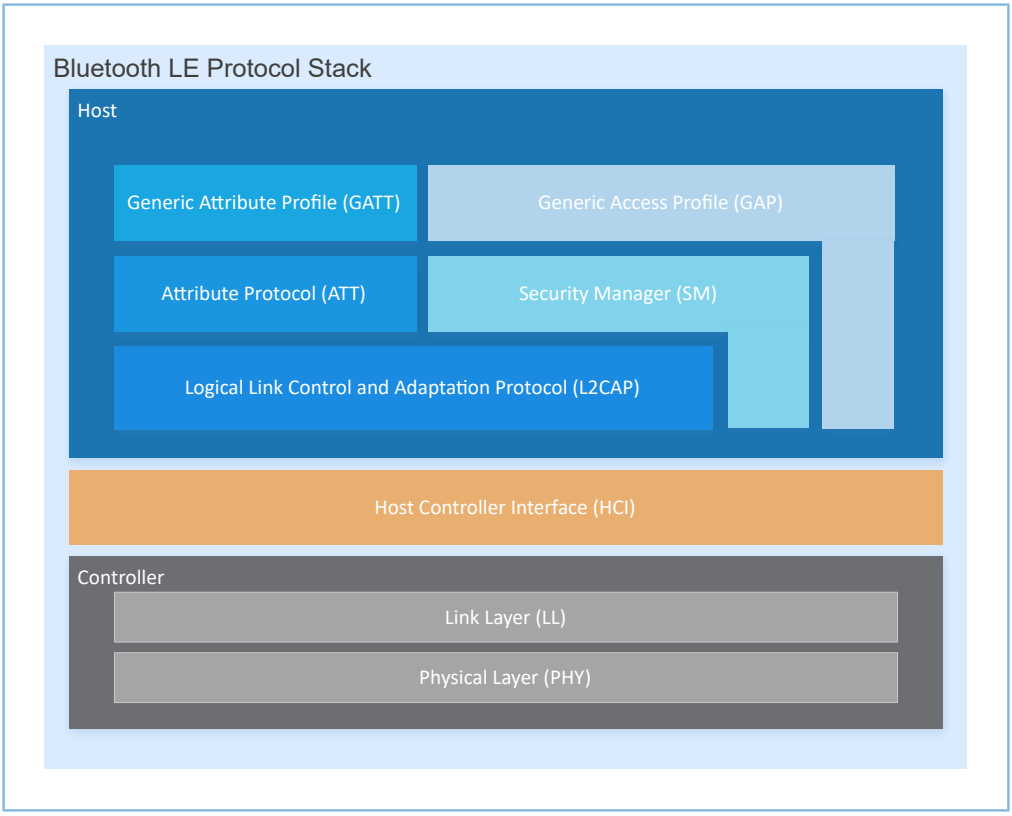

图 1-1 低功耗蓝牙协议栈架构

低功耗蓝牙协议栈由控制器（Controller）、主机控制接口（HCI）及主机（Host）组成。

控制器（Controller）

- 物理层（Physical Layer，PHY）：支持1 Mbps、2 Mbps、500 kbps及125 kbps的自适应跳频GFSK（高斯频移键控）射频（RF）操作。
- 链路层（Link Layer，LL）：控制设备的射频状态，支持五种设备状态（Standby、Advertising、Scanning、Initiating或Connection），并可根据实际需求切换状态。

主机控制接口（HCI）

- 主机控制接口（Host Controller Interface，HCI）：提供Host与Controller之间的通信接口。该接口层的实现可以是软件接口，也可以是标准硬件接口，例如UART、Secure Digital（SD）或USB。HCI commands和events通过该接口层在Host与Controller之间传递。

主机（Host）

- 逻辑链路控制和适配协议层（Logical Link Control and Adaptation Protocol，L2CAP）：为上层提供多路复用、数据分段与重组服务，并且支持逻辑端对端的数据通信。
- 安全管理层（Security Manager，SM）：定义配对和密钥分发的方法，为上层协议栈和应用程序提供端到端的安全连接和数据交换功能。
- 通用访问规范层（Generic Access Profile，GAP）：为上层应用和Profiles提供与协议栈通信交互的接口，主要用于实现广播、扫描、连接发起、服务发现、连接参数更新、安全过程发起与响应等功能。
- 属性协议层（Attribute Protocol，ATT）：定义了服务端和客户端之间的服务数据交互协议。
- 通用属性规范层（Generic Attribute Profile，GATT）：基于ATT协议之上，定义了一系列用于GATT Client与GATT Server之间服务数据交互的通信过程，供上层应用、Profile及Service使用。

提示:

- 更多Bluetooth LE技术及其协议的相关资料，请访问Bluetooth SIG的官方网站https://www.bluetooth.com。
- GAP、SM、L2CAP及GATT规范包含在Bluetooth Core Spec中，其他Bluetooth LE应用层的Profiles/Services规范可以在GATT Specs页面下载。Bluetooth LE应用可能会用到的Assigned Numbers、IDs及Codes均列在Assigned Numbers页面。

---

2 GR5405低功耗蓝牙软件平台

GR5405 SDK是基于GR5405芯片定义的低功耗蓝牙应用开发的软件套件，包括Bluetooth LE 5.3 API、System API及外设驱动API，并提供丰富的蓝牙和外设应用示例工程和使用说明文档。应用开发者可以基于GR5405 SDK的示例工程进行快速产品开发和迭代。

2.1 硬件架构

GR5405芯片的硬件框图，如下所示：

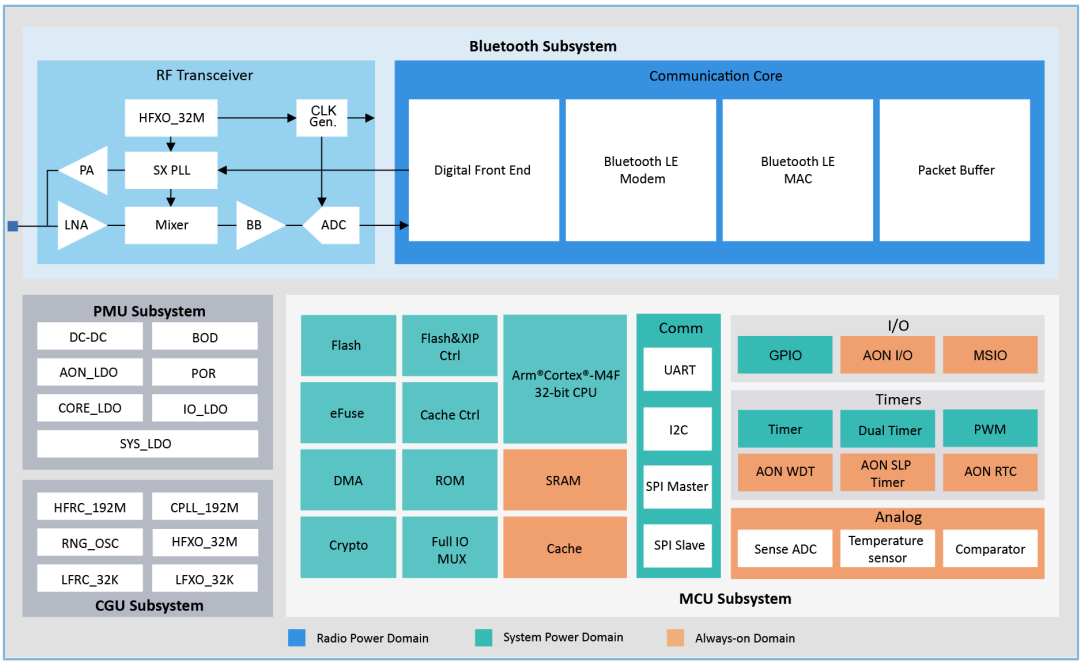

图 2-1 GR5405硬件框图

- Arm® Cortex®-M4F：GR5405 SoC芯片的系统核心处理器（CPU）。Bluetooth LE协议栈和Application代码均运行在该处理器上。
- SRAM：静态随机存取存储器，提供程序执行时所需的内存空间。
- ROM：只读存储器，固化了Bootloader和Bluetooth LE协议栈的软件代码。
- Flash：封装在芯片内部的Flash存储单元，用于存储用户代码和数据，支持用户代码片上运行模式（Execute in Place，XIP）。
- Peripherals：GPIO、DMA、I2C、SPI、UART、PWM、Timer、ADC及TRNG等硬件外设。
- RF Transceiver：2.4 GHz射频收发器。
- Communication Core：Bluetooth 5.3协议栈控制器的物理层，提供软件协议栈与2.4 GHz射频硬件之间的通信接口。
- PMU（Power Management Unit）：电源管理单元，为各系统模块提供电源电压。可根据配置参数和当前系统运行状态，设定合理的DC-DC、SYS_LDO、IO_LDO、CORE_LDO、RF Subsystem模块参数，实现功耗自动管理。

提示:

关于芯片各模块的详细介绍，请参考《GR5405 Datasheet》。

2.2 软件架构

GR5405 SDK的软件架构，如下图所示。

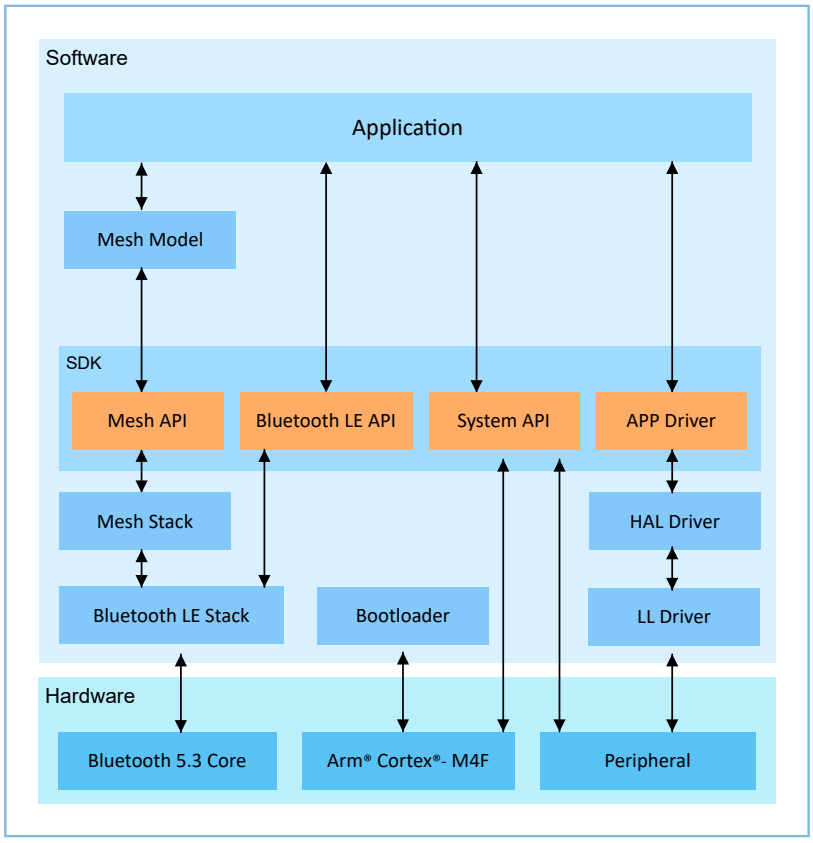

图 2-2 GR5405软件架构

- Bootloader

  固化在芯片中的引导程序，负责初始化芯片的软硬件环境，校验并启动应用程序。

- Bluetooth LE Stack

  低功耗蓝牙协议实现核心，由控制器（Controller）、主机控制接口（HCI）和主机（Host）协议组成（包括LL、HCI、L2CAP、GAP、SM、GATT），支持Broadcaster、Observer、Peripheral和Central角色。

- Mesh Stack

  Bluetooth LE Mesh协议实现核心，集成 Mesh 网络架构中的承载层、网络层、底层传输层、上层传输层、接入层以及部分基础模型的功能。

- LL Driver

  底层驱动层，直接利用寄存器操作驱动外设，包括对外设的控制和管理。

- HAL Driver

  硬件抽象驱动层，介于APP驱动层和LL驱动层之间的一个抽象层。其提供一组标准化的API接口，可方便APP驱动层通过调用HAL层API访问底层外设资源。

说明:

HAL层的API接口通常只适用于开发底层驱动和系统级服务，而不适用于普通应用程序开发。因此，不推荐开发者直接调用HAL层的API接口。

- Bluetooth LE SDK

  软件开发工具包，提供简单易用的Mesh API、Bluetooth LE API、System API以及APP Driver API。

  - Mesh API包括 Mesh 应用开发所需 API。
  - Bluetooth LE API包括L2CAP、GAP、SM和GATT API。
  - System API提供对非易失性数据存储系统（NVDS）、Firmware升级（DFU）、系统电源管理以及通用系统级访问的接口。
  - APP Driver为应用驱动层，基于HAL驱动将各外设的常用功能封装成一系列API。这些接口不仅具有HAL驱动的特性，而且更稳定、安全、易用。通常情况下，推荐开发者使用APP层API。

- Mesh Model

  Bluetooth SIG 定义的标准 Mesh Model（如Lightness Model等）的参考实现代码。开发者可基于这些参考示例代码开发Mesh Application。

- Application

  SDK包提供了丰富的蓝牙及外设示例工程，且每个示例工程都包含编译后的二进制文件，用户可以直接将其下载至芯片中运行和测试。对于大部分蓝牙应用，GRToolbox（Android）也提供了对应的功能，可方便用户测试。

2.3 存储器映射

GR5405系列SoC的存储器映射，如下图所示：

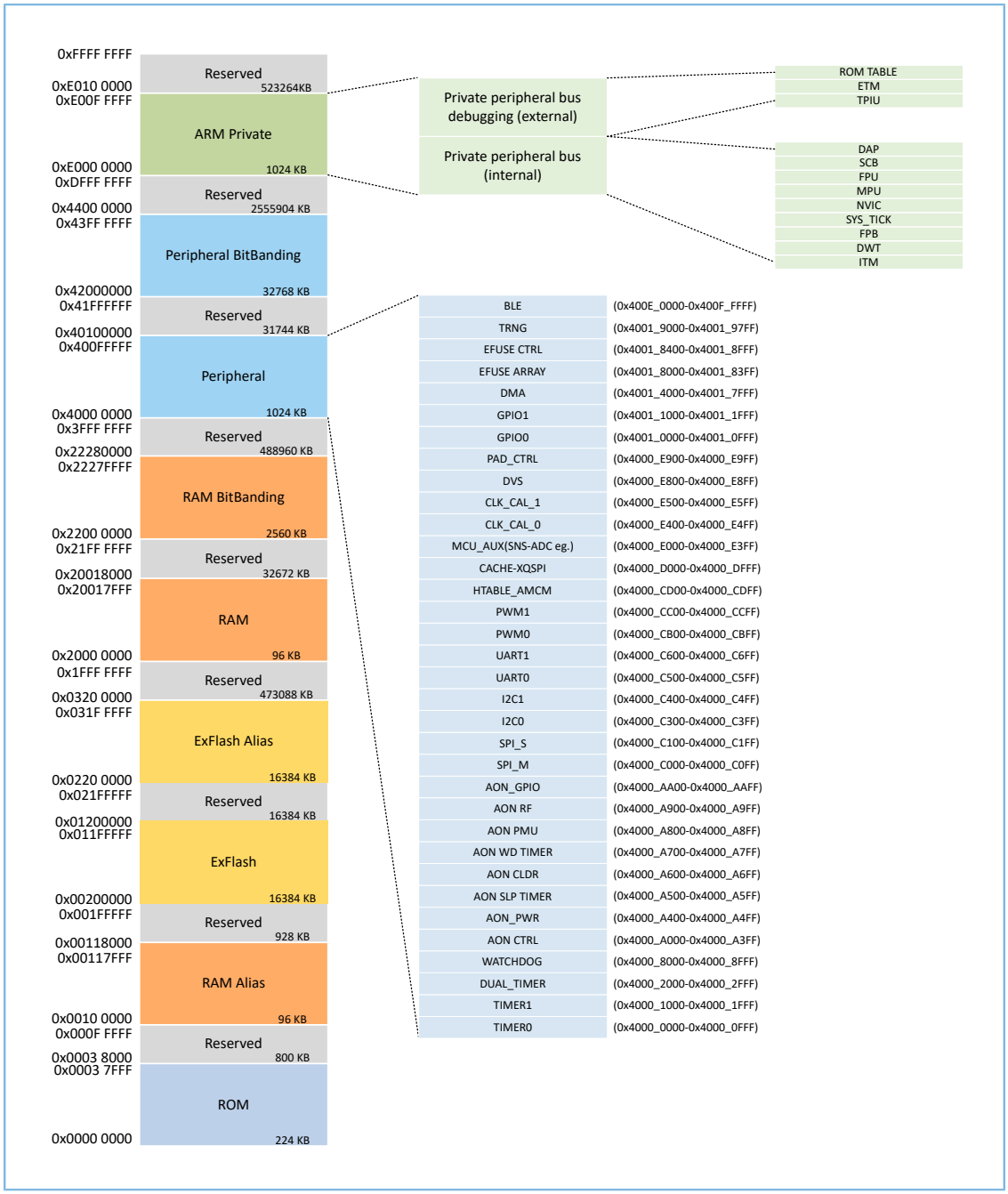

图 2-3 GR5405存储器映射

- RAM存储：共96 KB，地址为0x0010_0000 ～ 0x0011_7FFF或0x2000_0000 ～ 0x2001_7FFF。
  - 0x2000_0000 ～ 0x2001_7FFF：SDK中RW、ZI、HEAP、STACK等变量位于该区域。SRAM末端的16 KB存储区域，在配置Bluetooth LE工程时，可用作Baseband的EM（Exchange Memory）。EM实际使用区域由*custom_config.h*中配置的Bluetooth LE最大业务量决定，且未使用的EM区域将与其他SRAM区域形成连续的地址空间。另外，0x2000_0000 ～ 0x2001_3FFF区域支持位段操作，其对应的位段地址为0x2200_0000 ～ 0x2227_FFFF，可执行数据原子操作。
  - 0x0010_0000 ～ 0x0011_7FFF：基于Cortex®-M4F总线架构特点，该区域的访问效率高于其他区域。因此，SDK中RAM_CODE可执行代码位于该区域。

- Flash存储：GR5405芯片内部Flash为512 KB，地址为0x0020_0000 ～ 0x0027_FFFF。

2.4 Flash存储映射

GR5405封装了一个采用XQSPI总线接口的可擦除外部Flash存储器。该Flash物理上由若干个4 KB大小的Flash扇区（Sector）组成；逻辑上可根据不同的应用场景，划分为不同用途的存储区域。

为GR5405典型应用场景的Flash存储布局，如下图所示。

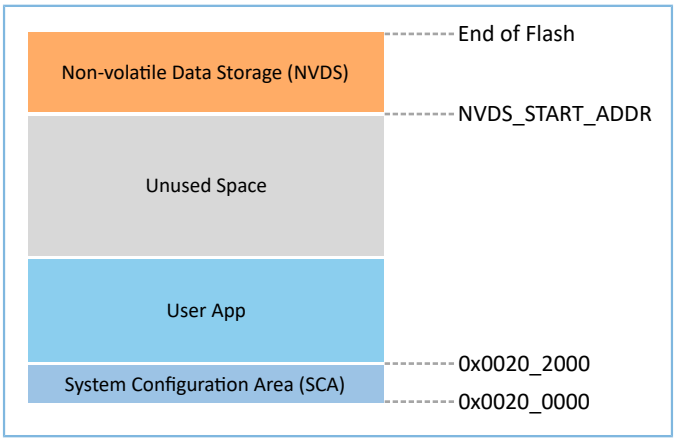

图 2-4 Flash存储布局

- System Configuration Area（SCA）：系统配置区，主要用于存储系统启动参数等配置信息。
- User App：Application Firmware存储区域，主要用于存储应用固件 。
- Unused Space：空闲区域，开发者可以自行使用该区域。例如，在DFU升级过程中，使用Unused Space临时存储新的Application Firmware。

说明:

- 开发者可以根据产品的Flash存储布局，合理配置NVDS的起始地址与所占用的Flash Sector数，具体配置方法参考4.3.2.1 配置custom_config.h。
- NVDS起始地址需与Flash Sector 起始地址对齐。
- 开发者可根据实际需求，自行移植littleFS等开源组件实现非易失性数据存储。

2.4.1 SCA

系统配置区（SCA）占用Flash前两个Sector（共8 KB，0x0020_0000 ～ 0x0020_2000），其主要存储系统启动过程使用的标志以及其他系统配置参数。

下载固件时，下载算法或GProgrammer工具会根据Application Firmware中的BUILD_IN_APP_INFO结构体生成 Image Info，并将其与应用固件一并烧写至Flash中（Image Info被存放在SCA中）。系统启动时，Bootloader根据SCA中的启动信息进行校验，校验通过后再跳转至Firmware的入口地址。

BUILD_IN_APP_INFO结构体的定义和配置如下：

提示:

该结构体位于SDK_Folder\platform\soc\common\gr_platform.c，其中SDK_Folder为GR5405 SDK包根目录。

```
const APP_INFO_t BUILD_IN_APP_INFO __attribute__((at(APP_INFO_ADDR))) = {
    .app_pattern = APP_INFO_PATTERN_VALUE,
    .app_info_version = APP_INFO_VERSION,
    .chip_ver = CHIP_VER,
    .load_addr = APP_CODE_LOAD_ADDR,
    .run_addr = APP_CODE_RUN_ADDR,
    .app_info_sum = CHECK_SUM,
    .check_img = BOOT_CHECK_IMAGE,
    .boot_delay = BOOT_LONG_TIME,
    .sec_cfg = SECURITY_CFG_VAL,
#ifdef APP_INFO_COMMENTS
    .comments = APP_INFO_COMMENTS,
#endif
.reserved1 = {APP_INFO_RESERVED}
};
```

- app_pattern：固定值0x47525858。
- app_info_version：固件信息版本，与APP_INFO_VERSION对应。
- chip_ver：固件对应的芯片版本，与*custom_config.h*中的CHIP_VER对应。
- load_addr：固件存储地址，与*custom_config.h*中的APP_CODE_LOAD_ADDR对应。
- run_addr：固件运行地址，与*custom_config.h*中的APP_CODE_RUN_ADDR对应。
- app_info_sum：固件信息的校验和，由软件自动计算CHECK_SUM。
- check_img：系统启动配置参数，与*custom_config.h*中的BOOT_CHECK_IMAGE对应。当此参数配置为“1”时，启动时Bootloader会对固件进行校验。
- boot_delay：启动配置参数，与*custom_config.h*中的BOOT_LONG_TIME对应。当此参数配置为“1”时，系统冷启动时将增加1秒延时。
- sec_cfg：安全配置参数，保留值。
- comments：固件描述信息，最大长度为12字节。

SCA的储存布局，如下图所示：

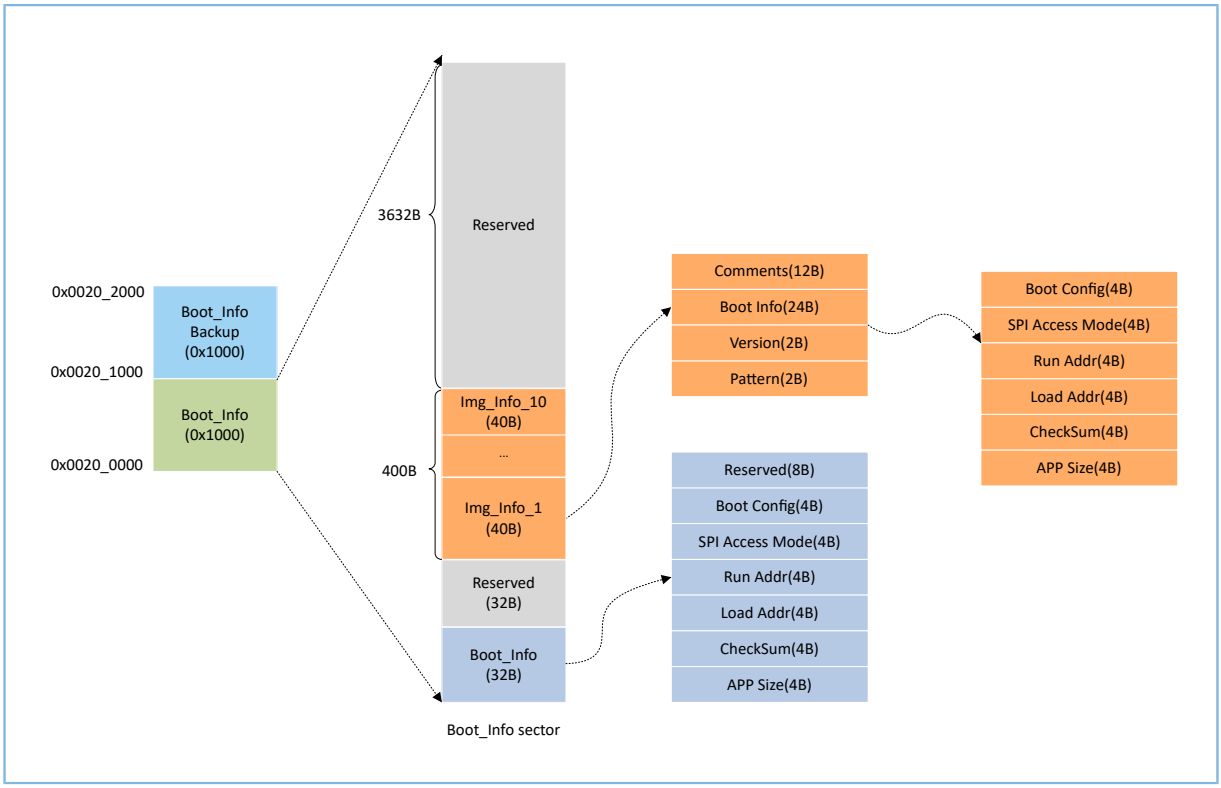

图 2-5 System Configuration Area布局

- Boot_Info与Boot_Info Backup存储相同信息，Boot_Info Backup为Boot_Info的备份。
- Boot_Info（32 B）区域中存储Firmware启动信息。系统启动时，Bootloader会根据启动信息进行校验，校验通过后跳转至Firmware的入口地址。
  - Boot Config：系统启动配置信息。
  - SPI Access Mode：SPI访问方式配置信息。系统固定配置，用户无法修改。
  - Run Addr：Firmware运行地址，与BUILD_IN_APP_INFO中的run_addr对应。
  - Load Addr：Firmware存储地址，与BUILD_IN_APP_INFO中的load_addr对应。
  - CheckSum：Firmware校验和，生成Firmware后，由下载算法自动计算。
  - APP Size：Firmware的Size信息，生成Firmware后，由下载算法自动计算。
- Img_Info区域可存储至多10个Firmware信息。当使用GProgrammer下载Firmware或使用DFU升级Firmware时，Firmware信息会被存储至Img_Info区域。
  - Comments：Firmware描述信息，最大长度为12个字符。生成Firmware后，下载算法使用Firmware文件名作为Comments信息。
  - Boot Info（24 B）：Firmware启动信息，与Boot_Info（32 B）的低24 Bytes数据相同。
  - Version：Firmware版本信息，与*custom_config.h*中VERSION对应。
  - Pattern：固定值0x4744。

2.4.2 NVDS

非易失性数据存储（Non-volatile Data Storage，NVDS）是一个轻量级键值对数据存储系统，其利用Flash硬件抽象层（Flash HAL）提供的Flash读写接口，将数据存储于Flash中，可确保系统掉电后数据也不会丢失。

NVDS系统适用于存储小块数据，如应用程序的配置参数、校准数据、状态信息以及用户信息等。低功耗蓝牙（Bluetooth LE）协议栈也会使用NVDS存储设备绑定等参数。

NVDS系统具有以下特性：

- 每个存储项（tag）都具有唯一的tag ID用于标识。用户程序可以根据tag ID对数据内容进行读取和更改操作，而无需关心数据存储的物理地址。
- 针对Flash存储介质的特性进行了优化，支持数据校验、垃圾回收和擦写均衡。
- 存储区域的大小和起始地址可配置。Flash区域以扇区（Sector）为单位，一个Sector的大小为4 KB。NVDS存储区域可配置为若干个Sector，且配置的起始地址需按4 KB对齐。

GR5405 SDK默认使用Flash最后几个Sector作为NVDS存储区域，其起始地址为Flash结束地址减去NVDS区域大小。开发者可在*custom_config.h*中配置宏NVDS_START_ADDR和NVDS_NUM_SECTOR来指定NVDS起始地址和Sector数。需注意NVDS_NUM_SECTOR不包含NVDS垃圾回收区域，整个NVDS所占用的Sector数为“NVDS_NUM_SECTOR + 1”。

说明:

关于NVDS的详细介绍和使用说明，请参考《GR54xx NVDS用户手册》。

2.5 RAM存储映射

GR5405 RAM起始地址为0x2000_0000，由6个内存块（RAM Block）组成（6*16 KB）。每个RAM内存块均可由软件独立打开或关闭电源。

96 KB RAM存储布局如下图所示。

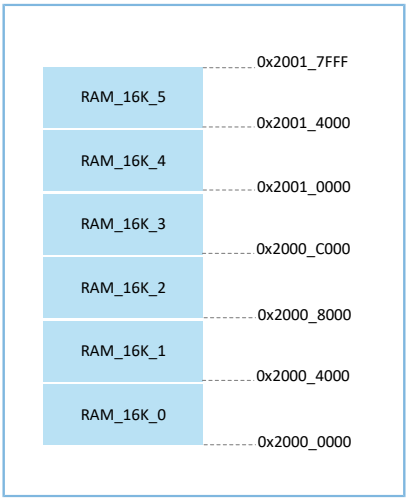

图 2-6 96 KB RAM存储布局

程序运行模式为Execute in Place（XIP）模式，即片上运行模式。用户应用程序存储于片上Flash空间，且程序运行空间和加载空间相同。系统完成上电配置后，通过Cache Controller直接从Flash空间取指运行。

2.5.1 典型RAM布局

下图为包含Bluetooth LE工程程序运行时的典型RAM布局。开发者可以根据产品需求，自定义RAM布局。

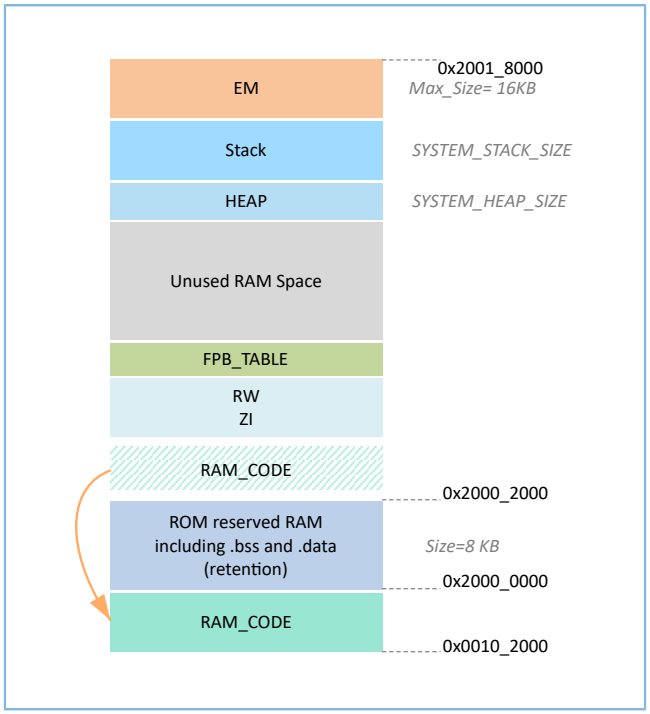

图 2-7 XIP模式的RAM（含Bluetooth LE工程）布局

- RAM CODE字段中存储了在RAM中执行的代码。为提高执行效率，建议将其定义在同物理地址的0x00100000 Aliasing Memory区域 。
- EM供Bluetooth LE CORE使用，并与MCU所使用的SRAM统一管理，位于SRAM的最高地址空间处。其Size取决于*custom_config.h*中配置的蓝牙业务数量。若工程不包含Bluetooth LE业务，则可将*custom_config.h*中宏BLE_SUPPORT的值设置为0。

- Stack字段中存储了任务调用栈。在不含Bluetooth LE业务的外设工程中，将Stack定义在RAM最高地址处；在含Bluetooth LE业务的工程中，将Stack定义在EM之下。Stack大小由宏SYSTEM_STACK_SIZE定义，开发者需根据工程的函数调用深度和调用栈消耗情况确定其大小。

2.5.2 RAM电源管理

每一个RAM Block可以处于三种不同电源状态：Full Power、Retention Power 或Power Off。

- Full Power：系统处于Active状态时，MCU可以进行RAM Block读写。
- Retention Power：系统进入Sleep模式时， RAM Block中存储的数据不会丢失，可供系统从Sleep状态恢复到Active状态态使用。
- Power off：系统关机时， RAM Block会掉电，其存储的数据也会丢失。因此，开发者需提前保存数据。

GR5405的电源管理单元（PMU）在系统启动时默认开启全部RAM电源。GR5405 SDK中也提供了完备的RAM电源管理API，开发者可以根据应用需求，合理配置RAM Block电源状态。

系统启动时，默认启用自动RAM功耗管理模式：根据Application的RAM使用情况，自动进行RAM Block电源状态控制，具体配置规则如下：

- 在系统Active状态下，将未使用的RAM Block设置为“Power off”状态，使用的RAM Block设置为“Full Power”状态。
- 当系统进入Sleep状态时，将未使用的RAM Block保持为“Power off”状态，而使用的RAM Block设置为“Retention Power”状态。

在实际应用中，建议RAM配置如下：

- 在Bluetooth LE应用中，RAM_16K_0的前 8 KB预留给Bootloader和Bluetooth LE协议栈使用，Application不可使用。在系统Active状态下，RAM_16K_0应处于“Full Power”状态；在系统Sleep期间，RAM_16K_0应处于“Retention Power”状态。非Bluetooth LE类MCU应用可以使用该RAM Block。
- RAM_16K_1及其他RAM Block的用途可由Application进行规划定义。GR5405 RAM已根据执行效率和SRAM的资源利用率进行合理布局，开发者也可根据实际应用需求重新规划。这些RAM Block的电源状态可以全部开启，也可以由Application自行控制。

说明:

- 仅当RAM Block处于“Full Power”状态时，MCU才能对其进行访问。
- 更多RAM电源管理API的详细说明，可参考`SDK_Folder\components\sdk\platform_sdk.h`。

2.6 SDK目录结构

GR5405 SDK的文件夹目录结构，如下所示：

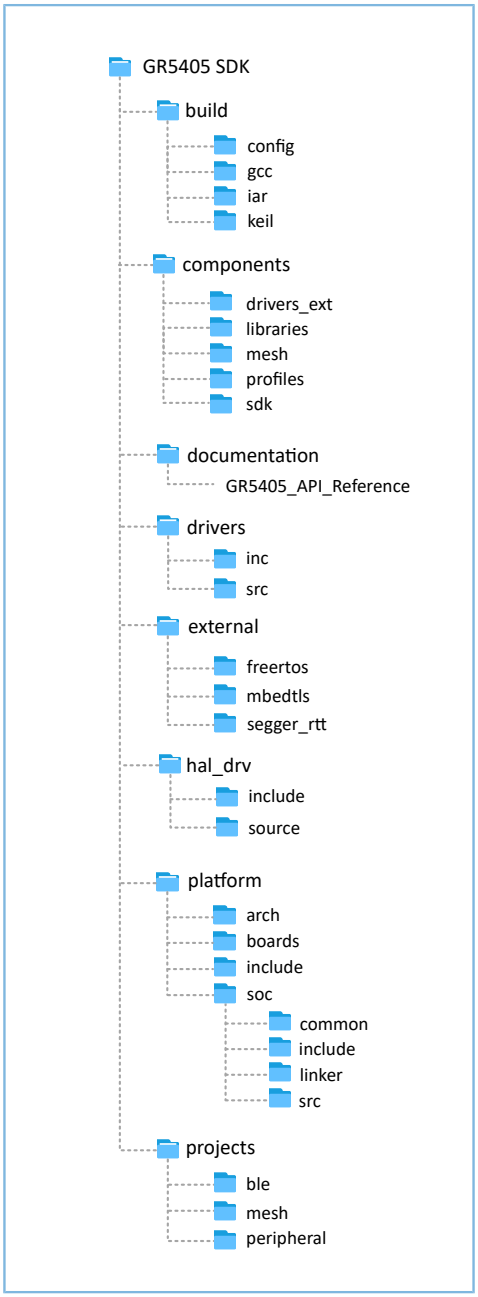

图 2-8 GR5405 SDK目录

GR5405 SDK中各文件夹的详细描述，参见下表：

表 2-1 GR5405 SDK文件夹

文件夹	描述
build\config	工程配置目录，用于存放*custom_config.h*模板文件。该文件主要用于配置工程参数。
build\gcc	运行GCC开发环境所需要使用的工具
build\keil	运行Keil开发环境所需使用的工具
build\iar	运行IAR开发环境所需要使用的工具
components\drivers_ext	开发板上第三方元器件的驱动
components\libraries	GR5405 SDK提供的libraries

文件夹	描述
components\profiles	GATT Services/Service Clients实现示例的源文件
components\mesh	Mesh API头文件、库文件，以及Mesh model实现的源文件
components\sdk	API头文件
documentation	GR5405 API Reference
drivers\inc	易于Application开发者使用的驱动API头文件
drivers\src	易于Application开发者使用的驱动API源代码
external\freertos	第三方程序，FreeRTOS源代码
external\mbedtls	第三方程序，Mbedtls源代码
external\segger_rtt	第三方程序，SEGGER RTT源代码
hal_drv\include	HAL驱动API头文件
hal_drv\source	HAL驱动API源文件
platform\arch	CMSIS的Toolchain文件
platform\boards	存放Starter Kit开发板的板级初始化源文件，主要实现对板级基础外设的初始化。
platform\include	存放与平台相关公共头文件
platform\soc\common	存放Goodix全系Bluetooth LE SoC兼容的公共源文件， 如*gr_interrupt.c*、*gr_platform.c*和*gr_system.c*
platform\soc\linker	链接器使用的符号表文件和库文件
platform\soc\include	存放SoC寄存器、时钟配置等与底层驱动强相关的公共头文件
platform\soc\src	存放*gr_soc.c*，主要实现SoC芯片强相关的一些初始化流程实现，如Flash、NVDS初始化，晶振配置和PMU校准等
projects\ble	Bluetooth LE Application工程示例，如ble_app_template
projects\mesh	Mesh Demo工程示例
projects\peripheral	芯片外设工程示例

2.7 配套工具

开发者可使用以下工具进行GR5405应用开发、调试：

表 2-2 开发/调试工具

名称	描述	推荐版本
GProgrammer	固件烧录工具，提供固件下载、Flash读写、eFuse烧写等功能，支持Windows/Linux平台。	V2.0.2及以上
GRUart	串口调试工具，仅支持Windows平台。	V2.1及以上
GRDirect Test Mode Tool	DTM测试工具，可通过发送HCI命令控制待测设备（DUT）执行测试，仅支持Windows平台。	V1.5.5及以上

名称	描述	推荐版本
GRPLT	量产配置工具，配合在线量产烧录板（PLT）使用，提供批量固件下载、Flash数据烧录、晶体校准以及多项测试功能，仅支持Windows平台。	V1.6.0.0.03及以上
GRCalibration	GR5xx专用蓝牙晶振校准工具，可用于校准GR551x系列、GR5405芯片及PLT量产烧录板的32 MHz晶振频偏，仅支持Windows平台。	V1.1.0及以上
GRToolbox	低功耗蓝牙功能调试手机APP，可扫描、配置连接参数、演示标准Profile等，提供Android/iOS版本。	V2.21及以上
GRMesh	用于配置、管理和控制Goodix蓝牙Mesh网络的手机APP，仅提供Android版本。	V1.07及以上

3 启动流程（Bootloader）

GR5405使用XIP（Execute in Place）模式运行代码。系统上电后，启动引导程序（Bootloader）从系统配置区（SCA）读取系统启动配置信息，并据此进行应用固件完整性校验、初始化Cache和XIP控制器后，再跳转至代码运行空间执行固件。

GR5405 SDK中的Application启动流程如下所示：

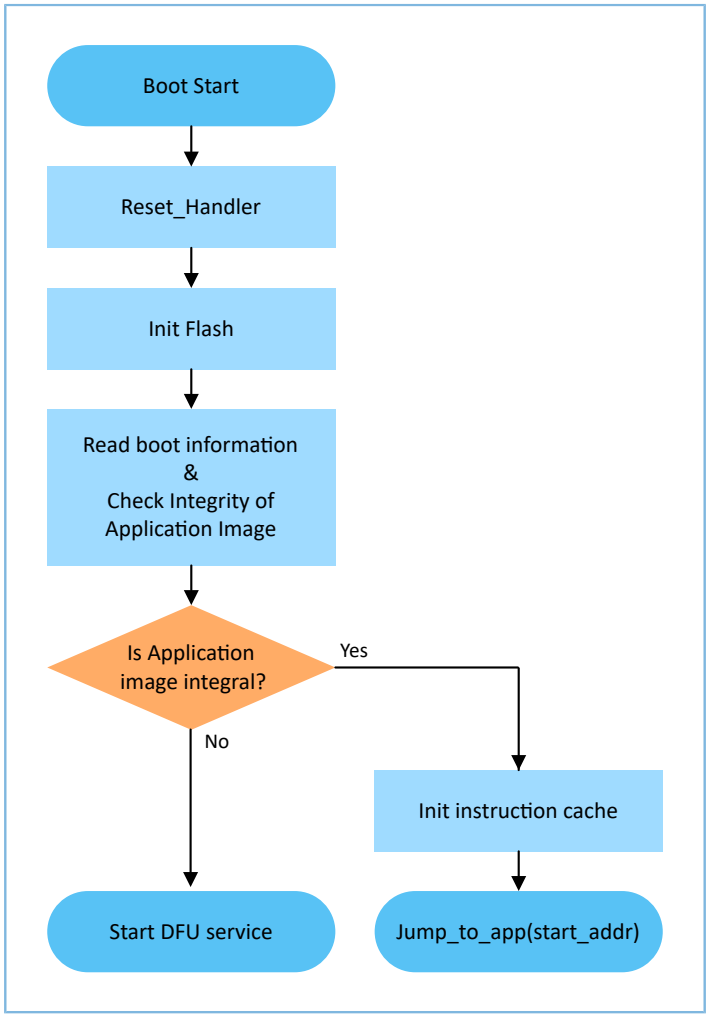

图 3-1 GR5405 SDK中的Application启动流程

1. 设备上电后，CPU将跳转至0x0000_0000处，从该地址处取出C-STACK栈顶指针并赋值给MSP，然后PC指向0x0000_004处，执行ROM中的Reset_Handler，进入Bootloader。
2. Bootloader执行Flash初始化。
3. Bootloader从Flash中的SCA读取启动信息，并执行Application Firmware的完整性检查。
4. 若Firmware完整性检查失败，则进入J-Link DFU模式。开发者可通过J-Link配合GProgrammer工具更新Flash中的Application Firmware。
5. 若Firmware完整性检查通过，则Bootloader完成XIP配置后将跳转至Flash中的Application Firmware的运行地址，开始执行代码。

4 使用SDK开发调试

本章将以Keil为例，介绍如何基于GR5405 SDK完成Bluetooth LE Application的创建、编译、下载以及调试。

4.1 安装Keil

Keil MDK-ARM IDE（Keil）是ARM®公司提供的用于Cortex®和ARM设备的集成开发环境（IDE）。开发者可从官方网站https://www.keil.com/demo/eval/arm.htm下载Keil安装包并进行安装。GR5405 SDK必须运行在Keil V5.20及以上的版本。

说明:

关于Keil MDK-ARM IDE的使用，可查看ARM提供的在线用户手册：https://www.keil.com/support/man_arm.htm。

图 4-1为Keil启动后的主界面。

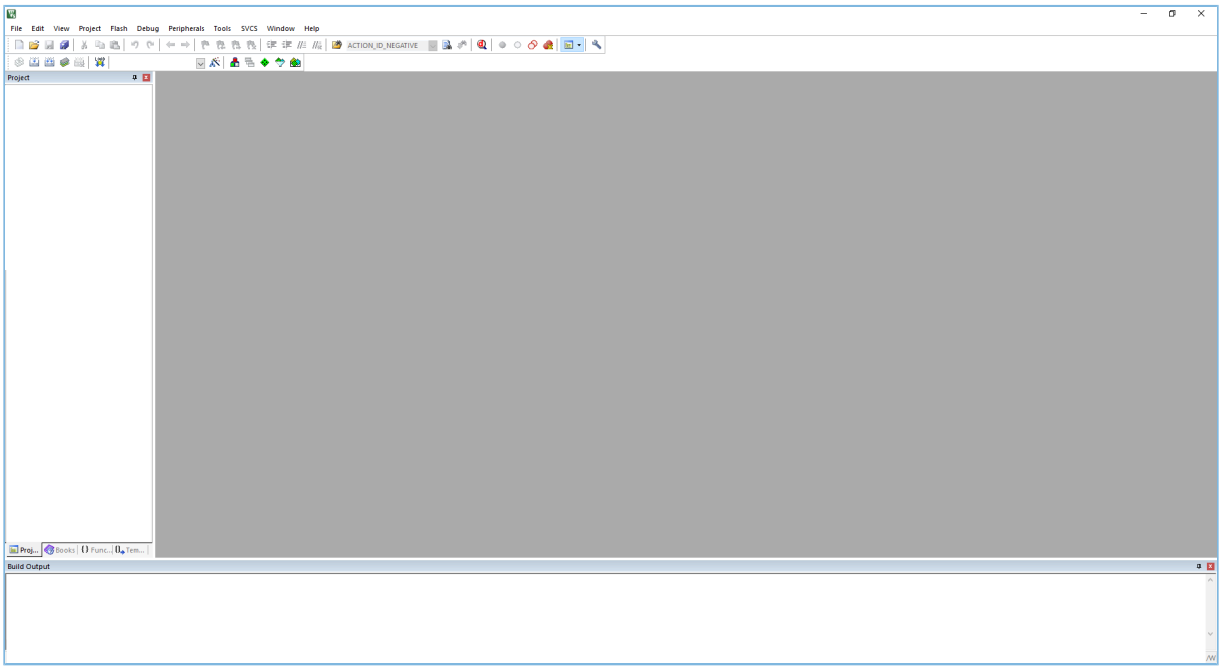

图 4-1 Keil软件界面

Keil的常用功能按钮包括：

表 4-1 Keil常用功能按钮

按钮	功能描述
	Options for Target
	Start/Stop debug session
	Download
	Build

4.2 安装SDK

GR5405 SDK包为.zip文件，直接解压即可使用。

说明:

- SDK_Folder为GR5405 SDK包的根目录。
- Keil_Folder为Keil的根目录。

4.3 创建Bluetooth LE Application

本节介绍基于Keil开发环境，利用GR5405 SDK快速创建用户自定义的Bluetooth LE应用。

4.3.1 准备ble_app_example

基于GR5405 SDK提供的模板工程，创建一个新工程。

进入`SDK_Folder\projects\ble\ble_peripheral\`，拷贝ble_app_template目录，并将其重命名为“ble_app_example”。将`ble_app_example\Keil_5`中的*.uvoptx*和*.uvprojx*的主文件名修改为“ble_app_example”。

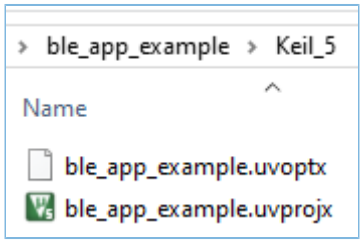

图 4-2 ble_app_example文件夹

双击*ble_app_example.uvprojx*，在Keil中打开工程。点击，打开“Options for Target 'GRxx_Soc'”窗口，选择“Output”标签页，在“Name of Executable”栏中输入“ble_app_example”，将生成的目标文件名称设置为“ble_app_example”。

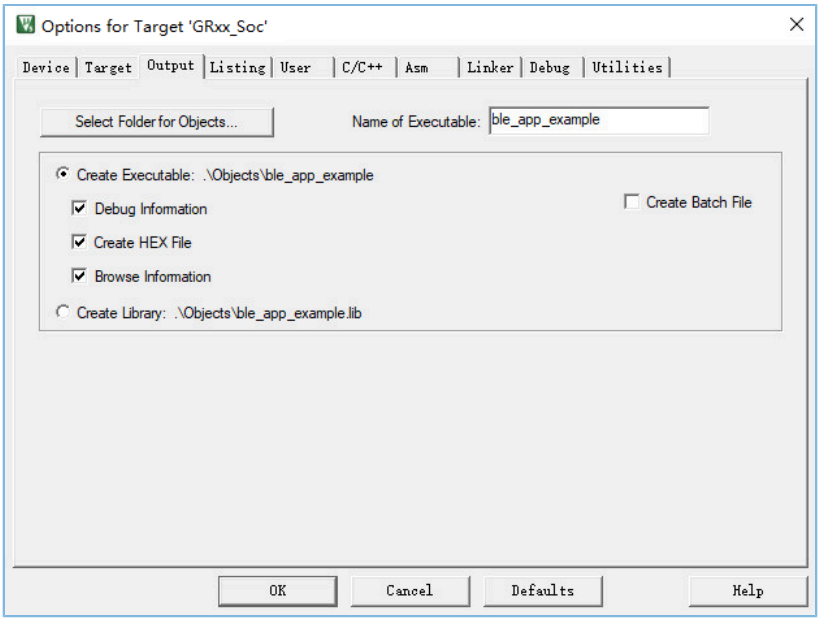

图 4-3 修改Name of Executable

在Keil Project Explorer中，可查看到ble_app_example工程下的所有groups。

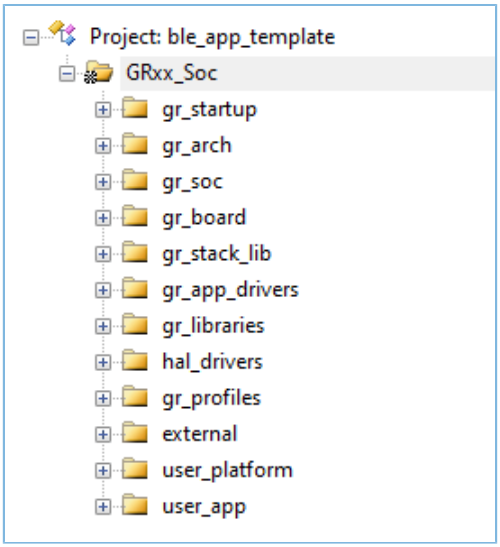

图 4-4 ble_app_example groups

ble_app_example工程下的groups主要分为两类：SDK groups和User groups。

- SDK group

  各SDK group的详细描述参见下表：

表 4-2 SDK groups

SDK group名称	描述
gr_startup	系统启动文件
gr_arch	System Core、PMU的初始化配置文件和系统中断的接口实现

SDK group名称	描述
gr_soc	与SoC相关的处理文件，主要是在进入main函数之前，对Clock、PMU、Vector等模块进行初始化与校准。
gr_board	板级描述文件，主要实现Log、Key、Led等相关组件。
gr_stack_lib	SDK lib文件
gr_app_drivers	易于Application开发者使用的驱动API源文件。开发者可根据实际需求，添加项目所需App 驱动。
gr_libraries	SDK提供的常用辅助软件模块、外设驱动的开源文件
hal_drivers	HAL驱动API源文件。开发者可根据实际需求，添加项目所需HAL 驱动。
gr_profiles	GATT Services/Service Clients源文件。开发者可根据实际需求，添加项目所需GATT源文件。
external	第三方程序的源文件，例如FreeRTOS、SEGGER RTT。开发者可根据实际需求，添加项目所需第三方程序。

- User groups

  各User group的详细描述参见下表：

表 4-3 User groups

User group名称	描述
user_platform	软硬件资源的配置和应用程序的初始化，开发者需要根据实际项目需求，实现相应接口。
user_app	主函数入口、开发者创建的其他源文件，配置Bluetooth LE协议栈运行时参数和实现GATT Service/Service Client的事件处理函数。

4.3.2 配置工程

开发者需根据产品特性，配置相应的工程选项，包括NVDS、代码运行模式、存储器布局以及其他配置项等。

4.3.2.1 配置custom_config.h

*custom_config.h*用于配置Application工程参数，开发者可选择直接修改文件或者使用Wizard界面进行配置。

提示:

各Application示例工程的*custom_config.h*位于其工程目录下的`Src\config`。

- 修改文件

  GR5405 SDK提供了一个Application工程配置模板文件（`SDK_Folder\build\config\custom_config.h`），开发者可直接修改该模板文件，配置Application工程参数。

表 4-4 custom_config.h中的参数

宏	描述
SOC_GR5405	定义芯片版本号。
CHIP_TYPE	芯片型号 0：GR5405 **说明：** 工程编译时，应根据实际使用的芯片型号配置该参数。
CFG_APP_DRIVER_SUPPORT	是否使用外设APP驱动。 0：不使用 1：使用
ENABLE_BACKTRACE_FEA	使能/禁用栈回溯功能。 0：禁用栈回溯功能 1：使能栈回溯功能
APP_LOG_ENABLE	APP LOG模块开关 0：禁用Application Log模块 1：使能Application Log模块
APP_LOG_STORE_ENABLE	APP LOG STORE模块开关 0：禁用APP LOG STORE模块 1：使能APP LOG STORE模块
APP_LOG_PORT	设置APP LOG输出方式。 0：UART 1：J-Link RTT 2：ARM ITM
PLATFORM_SDK_INIT_ENABLE	使能/禁用Platform初始化。 0：禁用Platform初始化 1：使能Platform初始化
PMU_CALIBRATION_ENABLE	使能/禁用PMU校准功能。 使能该功能后，系统将自动监测温度与电压，并自适应调整。建议默认使能。 0：禁用PMU校准 1：使能PMU校准 **说明：** 在高/低温使用场景下，必须使能PMU校准功能。
NVDS_START_ADDR	NVDS占用Flash区域的起始地址，默认为0。 如果未指定地址，NVDS默认使用Flash末尾区域。

宏	描述
NVDS_NUM_SECTOR	NVDS占用的Flash Sector数，默认为3。
SYSTEM_STACK_SIZE	Application所需的Call Stack的大小，默认为8 KB。 开发者可根据实际使用情况，调整该值 。 **说明：** ble_app_example示例工程编译后，可在其目录下的`Keil_5\Objects\ble_app_example.htm`文件中查看参考的Maximum Stack Usage。
SYSTEM_HEAP_SIZE	Application所需的Heap大小，默认为0 KB。 开发者可根据实际使用情况，调整该值。
CHIP_VER	固件使用的芯片版本，默认值为0x5405。
APP_CODE_LOAD_ADDR *	程序存储空间的起始地址 **说明：** 该地址需在Flash地址范围内。
APP_CODE_RUN_ADDR *	程序运行空间的起始地址 **说明：** 该地址值需与APP_CODE_LOAD_ADDR保持一致，Application采用XIP模式运行。
SYSTEM_CLOCK *	设置系统时钟频率。 0：64 MHz 1：32 MHz 2：16 MHz（XO） 3：16 MHz 4：8 MHz 5：2 MHz
RF_TX_PA_SELECT	射频功率放大器选择 1：SPA（支持-20 dBm ～ 5 dBm TX功率） 2：HPA（支持-10 dBm ～ 15 dBm TX功率）
SYSTEM_POWER_MODE	设置系统供电方式。 0：采用DC-DC供电 1：采用SYS_LDO供电 当射频功率放大器使用HPA时，系统采用SYS_LDO供电。
CFG_LF_ACCURACY_PPM	Bluetooth LE低频睡眠时钟精度，其取值范围为1 ～ 500，单位PPM。
CFG_LPCLK_INTERNAL_EN	是否使用芯片内部的OSC时钟作为Bluetooth LE低频睡眠时钟。 若使用，则CFG_LF_ACCURACY_PPM将被强制设置为500 PPM 。 0：不使用 1：使用

宏	描述
BOOT_LONG_TIME *	设置芯片启动时是否需要延迟1s再执行后半段启动代码。 0：不延迟 1：延迟1秒
BOOT_CHECK_IMAGE	在XIP模式下，冷启动时是否对Image进行校验。 0：不进行校验 1：进行校验
BLE_SUPPORT	是否支持Bluetooth LE。 0：无Bluetooth LE，仅支持MCU 1：支持
CFG_CONTROLLER_ONLY	是否只支持 Bluetooth LE 控制器（用于外部主机或 HCI uart 传输）。 0：支持 Bluetooth LE 控制器和主机 1：仅支持 Bluetooth LE 控制器
CFG_BT_BREDR	是否支持LE链路生成Bluetooth Classic Link Key。 0：不支持 1：支持
DTM_TEST_V1_CMD_ENABLE	使能/禁用DTM测试v1命令。 0：禁用DTM测试v1命令 1：使能DTM测试v1命令
DTM_TEST_V2_CMD_ENABLE	使能/禁用DTM测试v2命令。 0：禁用DTM测试v2命令 1：使能DTM测试v2命令
DTM_TEST_V3_CMD_ENABLE	使能/禁用DTM测试v3命令。 0：禁用DTM测试v3命令 1：使能DTM测试v3命令
DTM_TEST_V4_CMD_ENABLE	使能/禁用DTM测试v4命令。 0：禁用DTM测试v4命令 1：使能DTM测试v4命令
DTM_TEST_PRIVATE_CMD_ENABLE	使能/禁用DTM测试私有命令。 0：禁用DTM测试私有命令 1：使能DTM测试私有命令
CFG_MAX_PRFS	支持的最大GATT Profile/Service数 开发者可根据实际需求，设置该值。值越大，占用的RAM空间越大。 取值范围：1 ～ 64
CFG_MAX_BOND_DEVS	支持的最大绑定设备数，默认值为4，最小值为1。
CFG_MAX_CONNECTIONS	支持的最大连接数，最大值为8。

宏	描述
	开发者可根据实际需求，设置该值。值越大，Bluetooth LE Stack Heap占用的RAM空间就越大。 Bluetooth LE Stack Heap具体大小由以下四个宏（位于*flash_scatter_config.h*）定义： ◦ ENV_HEAP_SIZE ◦ ATT_DB_HEAP_SIZE ◦ KE_MSG_HEAP_SIZE ◦ NON_RET_HEAP_SIZE **说明：** 开发者不可修改上述四个宏。
CFG_MAX_ADVS	支持的最大Bluetooth LE Legacy/Extended Advertising 数，取值范围：0 ～ 4。 **注意：** Bluetooth LE 广播（Legacy/Extended/Periodic Advertisings）总数不应超过 4。
CFG_MAX_SCAN	支持的最大 Bluetooth LE 扫描数，取值范围：0 ～ 1。
CFG_MUL_LINK_WITH_SAME_DEV	是否支持同一设备连接多个从设备。 0：不支持 1：支持
CFG_CCC_SC_OOB_PAIR_SUPPORT	是否支持CCC 3.0 Bluetooth LE OOB安全配对。 0：不支持 1：支持
CFG_MASTER_SUPPORT	是否支持Master角色。 0：不支持 1：支持
CFG_SLAVE_SUPPORT	是否支持Slave角色。 0：不支持 1：支持
CFG_SUPPER_ADV_SUPPORT	是否支持低占空比广播的最小间隔为5 ms。 0：不支持，按照规格为20 ms 1：支持
CFG_LEGACY_PAIR_SUPPORT	是否支持Legacy配对。 0：不支持 1：支持
CFG_SC_PAIR_SUPPORT	是否支持Secure配对。 0：不支持 1：支持

宏	描述
CFG_COC_SUPPORT	是否支持面向连接通道（COC）。 0：不支持 1：支持
CFG_GATTS_SUPPORT	是否支持GATT Server。 0：不支持 1：支持
CFG_GATTC_SUPPORT	是否支持GATT Client。 0：不支持 1：支持
CFG_CONN_AOA_AOD_SUPPORT	是否支持基于连接AoA/AoD。 0：不支持（默认） 1：支持 **说明：** 该宏固定配置为0。
CFG_CONNLESS_AOA_AOD_SUPPORT	是否支持无连接AoA/AoD。 0：不支持 1：支持 **说明：** 该宏固定配置为0。
CFG_MESH_SUPPORT	是否支持Mesh功能。 0：不支持 1：支持
SECURITY_CFG_VAL	安全配置 0： 启用一级算法 1： 启用二级算法
WDT_RUN_ENABLE	设置WDT功能。 0：禁用后台运行WDT 1：使能后台运行WDT

说明:

*：上表中带*的宏可用于初始化BUILD_IN_APP_INFO结构体，该结构体被定义在固件的0x200地址处，并使用*custom_config.h*中的宏进行初始化。系统启动时，Bootloader程序会从该地址读取固件的配置信息，作为启动参数。

- 使用Wizard配置参数

  *custom_config.h*文件中的注释符合Keil的ConfigurationWizard Annotations规范。因此，开发者还可以在Keil中打开“Configuration Wizard”工具，配置*custom_config.h*中的宏。

提示:

推荐开发者使用Wizard进行参数配置，以避免出现非法参数值。

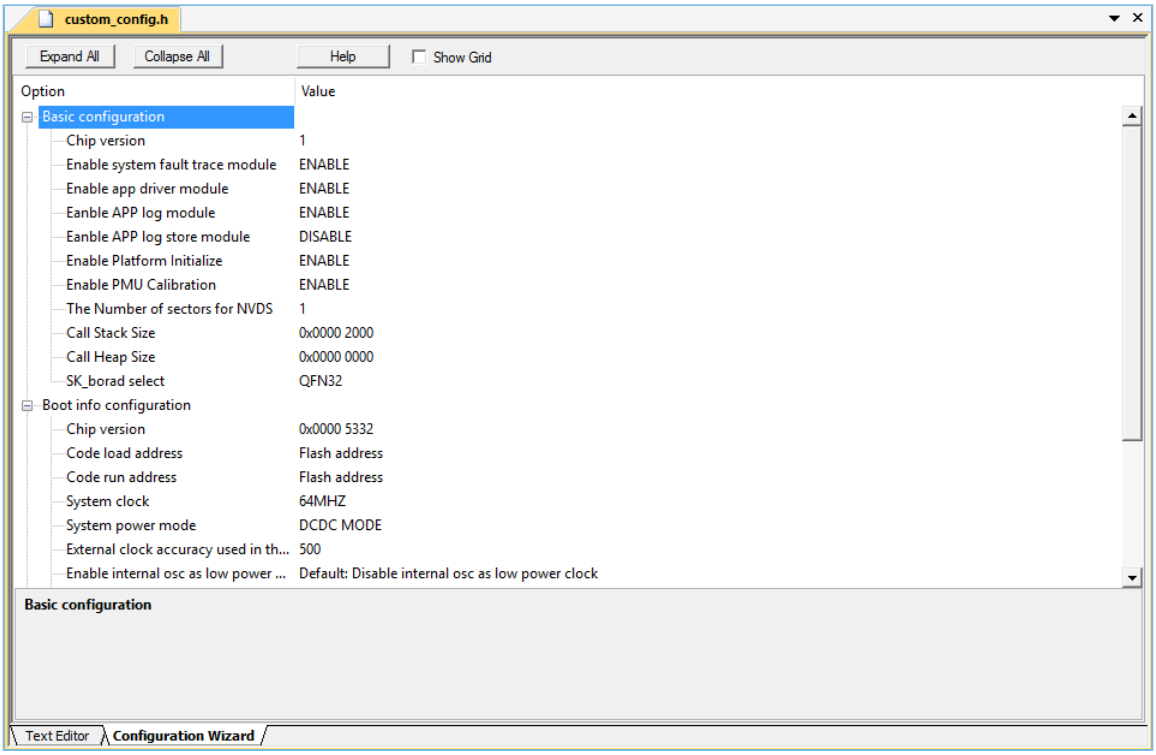

图 4-5 Configuration Wizard for custom_config.h

4.3.2.2 配置存储器布局

在Keil工程中，Scatter （.sct）文件描述链接器使用的存储区域。GR5405 SDK提供了一个Scatter示例文件（`SDK_Folder\platform\soc\linker\keil\flash_scatter_common.sct`），可帮助开发者快速完成存储器布局配置。另外，*flash_scatter_common.sct*使用的宏定义在*flash_scatter_config.h*中。

说明:

在Keil中，`__attribute__((section("name")))`可用于将一个函数或变量定义在特定的内存段中，其中“name”由开发者自定义。Scatter(*.sct*)文件可用于将自定义字段定义在特定位置。例如，将应用程序的ZI（零初始化）数据定义在名称为“.bss.app”的内存段中，则可设置“attribute”为“attribute ((section(".bss.app ")))”。

开发者可按照以下步骤配置存储器布局：

1. 点击Keil 工具栏中的“Options for Target”按钮，打开“Options for Target ‘GRxx_Soc’ ”对话框，再选中“Linker”标签页。

2. 在“Linker”页面中的“Scatter File”栏，点击按钮“...”，浏览选择SDK_Folder\platform\soc\linker\keil下的*flash_scatter_common.sct*文件。开发者还可先将Scatter文件（.sct）和配置文件（.h）拷贝至ble_app_example工程的对应目录，再浏览选择文件。

说明:

- *flash_scatter_common.sct*中的#! armcc -E –I语句指定了*flash_scatter_common.sct*依赖的头文件的目录。若该路径错误，则会产生Linker Error。
- 在GR5405 SDK的Scatter文件*flash_scatter_common.sct*中，可利用*custom_config.h*的宏定义BLE_SUPPORT确定SRAM末端区域是否需要配置用于Bluetooth LE的EM。开发者需根据工程是否包含Bluetooth LE业务，对BLE_SUPPORT进行定义。

---

3. 点击“Edit...”按钮，打开.sct文件，然后根据实际的产品存储器布局，修改相应代码。

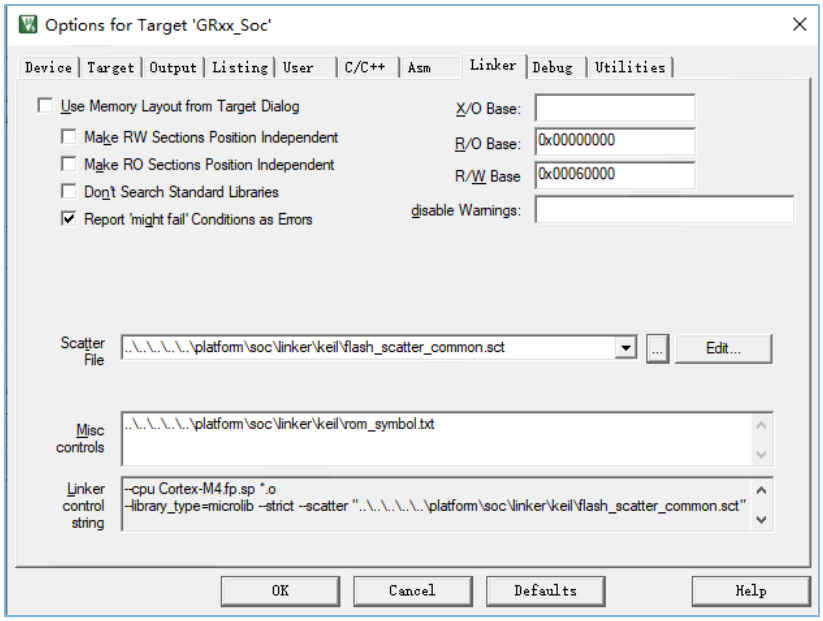

图 4-6 配置Scatter File

4. 点击“OK”按钮，保存设置。

4.3.3 添加用户代码

开发者可根据实际应用需求，修改ble_app_example中相应代码。

4.3.3.1 修改主函数

以下为典型的*main.c*文件内容：

```
/**@brief Stack global variables for Bluetooth protocol stack. */
 STACK_HEAP_INIT(heaps_table);
...
int main (void)
{
```

```
    // Initialize user peripherals.
    app_periph_init();

    // Initialize ble stack.
    ble_stack_init(ble_evt_handler, &heaps_table);

    // loop
    while (1)
    {
        app_log_flush();
        pwr_mgmt_schedule();
    }
}
```

- `STACK_HEAP_INIT(heaps_table)`定义了7个全局数组，供Bluetooth LE协议栈作为Heap使用。开发者不可修改此定义，否则Bluetooth LE协议栈可能无法正常运行。Heap大小与4.3.2.1 配置custom_config.h中的Bluetooth LE业务量有关。
- `app_periph_init()`用于初始化外设。在开发调试阶段，该函数中的SYS_SET_BD_ADDR可用于设置临时的Public Address，pwr_mgmt_mode_set()中可设置在自动功耗管理时MCU的工作模式（SLEEP/IDLE/ACTIVE）。app_periph_init()函数实现位于*user_periph_setup.c*文件，其示例代码如下所示：

```
/**@brief Bluetooth device address. */
static const uint8_t s_bd_addr[SYS_BD_ADDR_LEN] = {0x11, 0x11, 0x11, 0x11,0x11, 0x11};
...
void app_periph_init(void)
{
    SYS_SET_BD_ADDR(s_bd_addr);
    board_init();
    pwr_mgmt_mode_set(PMR_MGMT_SLEEP_MODE);
}
```

- 在`while(1) { }`中添加Application的Main Loop代码，比如处理外部输入、更新GUI。
- 若需打开App Log模块，则应在Main Loop中调用`app_log_flush()`，以保证系统进入Sleep状态之前，完整输出所有Log。关于App Log模块使用，参考4.6.3 输出调试Log。
- 调用`pwr_mgmt_shcedule()`实现自动功耗管理，以降低系统功耗。

4.3.3.2 实现Bluetooth LE业务逻辑

Application的Bluetooth LE业务逻辑由GR5405 SDK中定义的若干Bluetooth LE Events进行驱动。因此，Application需要实现相应的Event Handler，以获取Bluetooth LE Stack的运行结果或状态变更通知。由于Event Handler是在Bluetooth LE SDK IRQ的中断上下文（Interrupt Context）中被调用的，因此开发者不能在Handler中执行比较耗时的操作，例如阻塞式函数调用、无限循环等。否则，可能阻塞整个系统运行，导致Bluetooth LE Stack与SDK Bluetooth LE模块无法按照正常时序运行。

Bluetooth LE Events按照Common、GAP Management 、GAP Connection Control、Security Manager 、L2CAP、GATT Common、GATT Server和GATT Client分类。

说明:

GR5405 SDK支持的Bluetooth LE Events可在SDK_Folder\components\sdk\ble_event.h中查看。

开发者需根据产品的功能需求，实现所需的Bluetooth LE Event Handler。例如，若产品不支持Security Manager，则可无需实现对应的Event；若产品只支持GATT Server而不支持GATT Client，则可无需实现GATT Client对应的Event。并且，对于每类Event的Event Hander也并非需全部实现，仅需实现产品所必须的Event Hander即可。

提示:

关于Bluetooth LE API和Event API的使用方法，请参考SDK_Folder\documentation\GR5405_API_Reference以及SDK_Folder\projects\ble中的Bluetooth LE示例源代码。

4.3.3.3 BLE_Stack_IRQ、BLE_SDK_IRQ与Application的调度机制

Bluetooth LE Stack是低功耗蓝牙协议实现核心，可直接操作Bluetooth 5.3 Core硬件（参考2.2 软件架构）。因此，BLE_Stack_IRQ具有整个系统中次高的优先级（SVCall IRQ具有最高优先级），以保证Bluetooth LE Stack严格按照Bluetooth Core Spec规定的时序运行。

Bluetooth LE Stack的状态改变会触发优先级较低的BLE_SDK_IRQ中断。在该中断处理函数中，可通过调用Application实现的Bluetooth LE Event Handler，将Bluetooth LE Stack的状态变更通知以及相关的业务数据发送至Application。在这些Event Handler中，应避免操作耗时的业务，而应将耗时业务转移至Main Loop或用户级线程中处理。开发者可使用SDK_Folder\components\libraries\app_queue模块（或自定义的Application Framework）从Bluetooth LE Event Handler向Main Loop传递事件。

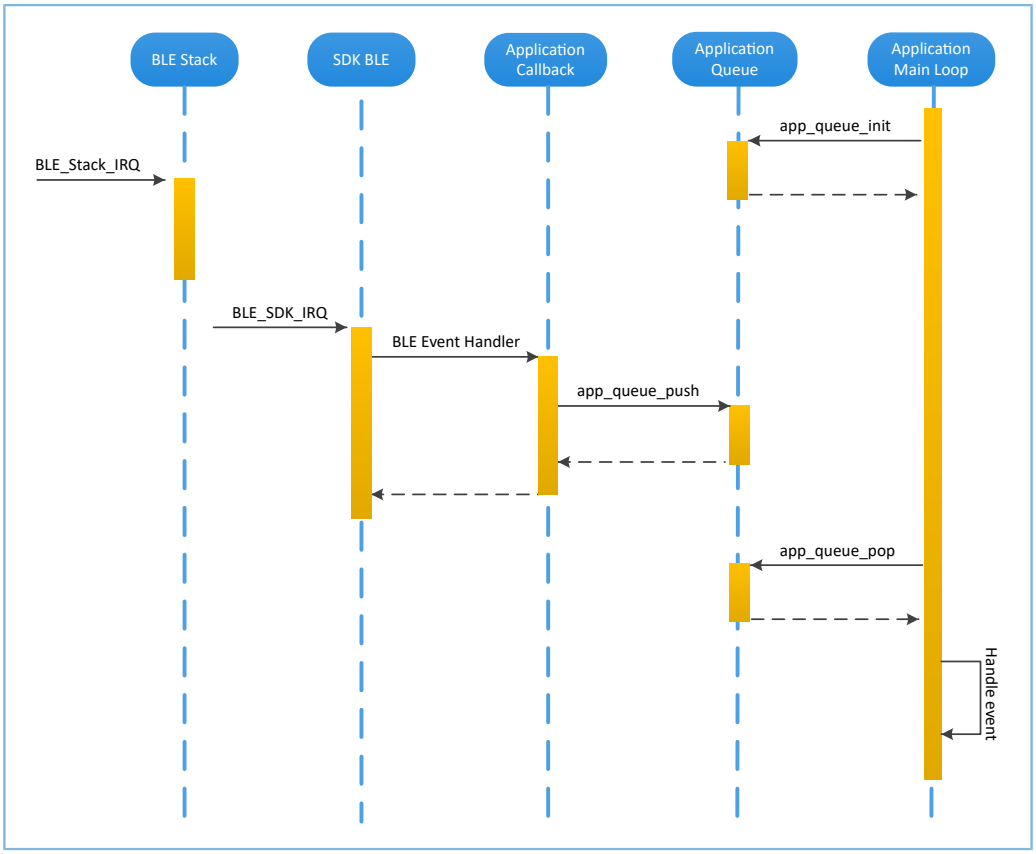

图 4-7 Non-OS system schedule

4.4 生成固件

Bluetooth LE Application创建完成以后，可直接点击Keil 工具栏中的“Build” 按钮构建工程。

工程编译成功后，将在工程目录下的`Listings`和`Objects`文件夹下分别生成.bin和.hex格式的固件文件。

表 4-5 生成的固件

名称	描述
ble_app_example.bin	二进制应用固件，可通过GProgrammer下载至芯片Flash中运行。
ble_app_example.hex	二进制应用固件，可通过Keil或GProgrammer下载至芯片Flash中运行。

提示:

上述两种格式的固件，均可使用GProgrammer下载至芯片Flash中运行。具体操作，可参考《GProgrammer用户手册》。

4.5 下载.hex文件至Flash

固件生成后，可按以下步骤将固件下载至Flash中：

1. 配置Keil Flash编程算法。

(1) 拷贝SDK_Folder\build\Keil\GR5xxx_16MB_Flash.FLM文件至Keil_Folder\ARM\Flash目录。

(2) 点击Keil 工具栏中的“Options for Target”按钮，打开“Options for Target ‘GRxx_Soc’”对话框，选择“Debug”标签页；点击“Use: J-LINK/J-TRACE Cortex”右侧的“Settings”按钮。

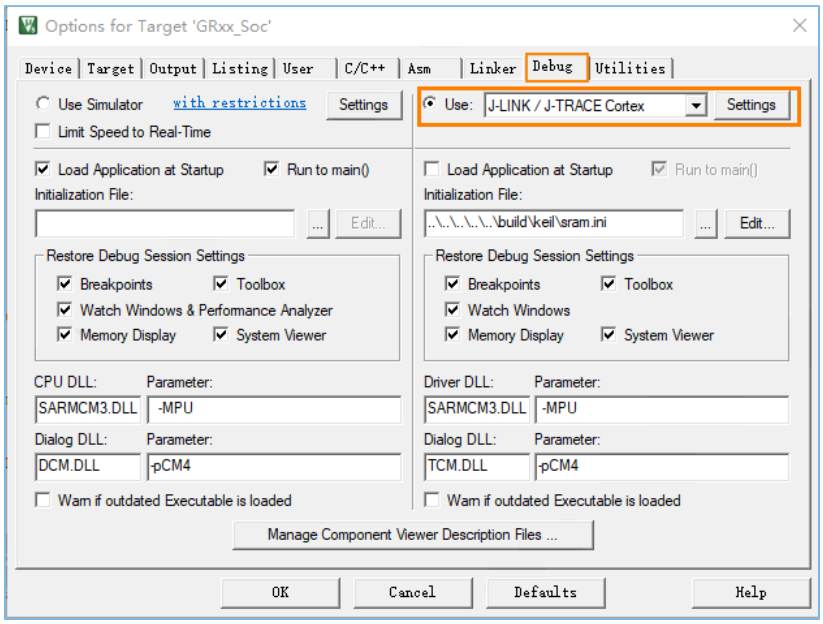

图 4-8 Debug标签页

(3) 在打开的“Cortex JLink/JTrace Target Driver Setup”窗口中，选中“Flash Download”项。在“Download Function”区域，开发者可以设置Erase方式、选择是否“Program”、“Verify”、“Reset and Run”。Keil默认配置如下：

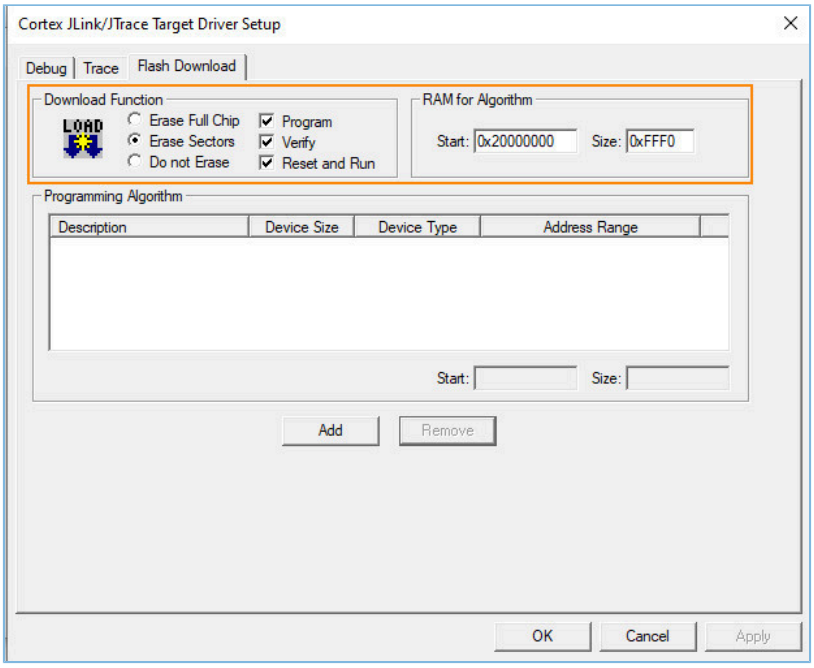

图 4-9 选择“Download Function”

(4) 点击“Add”按钮，在“Programming Algorithm”中添加SDK_Folder\build\keil\GR5xxx_16MB_Flash.FLM。

说明:

为方便用户多芯片继承性开发，合并GR5xx系列Bluetooth LE芯片下载算法，均使用*GR5xxx_16MB_Flash.FLM*文件。

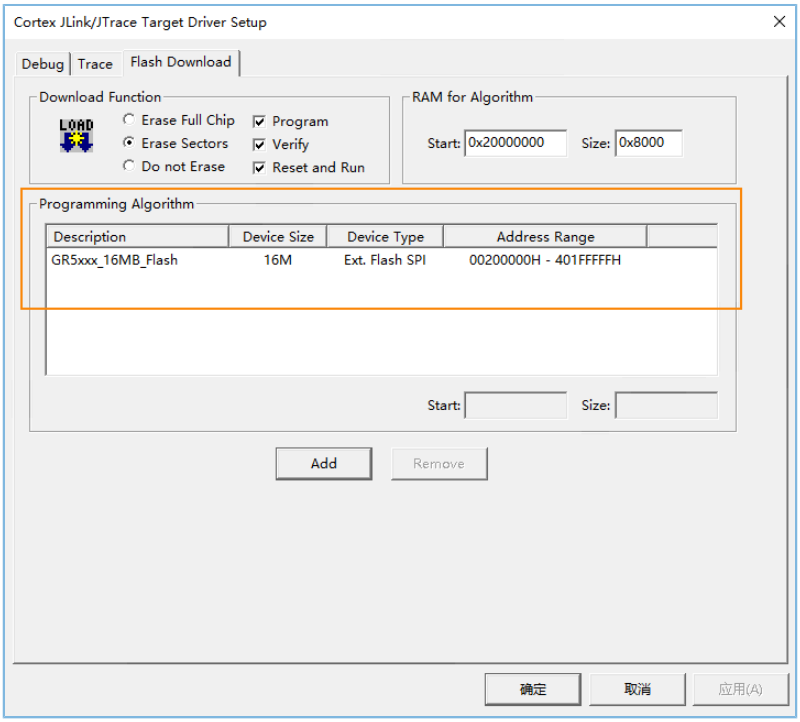

图 4-10 添加GR5xxx_ 16MB_Flash.FLM编程算法

(5) 配置“RAM for Algorithm”，以定义加载和执行编程算法的地址空间。“Start”值应为GR5405 RAM的起始地址“0x20000000”，“Size”值为“0x8000”。

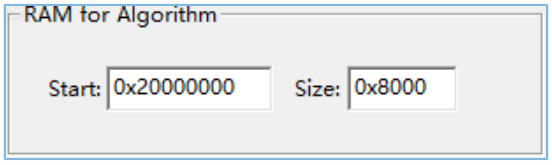

图 4-11 “RAM for Algorithm”设置

(6) 点击“OK”，保存设置。

2. 下载固件。

配置完成以后，点击Keil 工具栏中的“Download”按钮 将*ble_app_example.axf*文件下载至芯片Flash中。如果固件下载成功，Keil的“Build Output”窗口将显示如下结果。

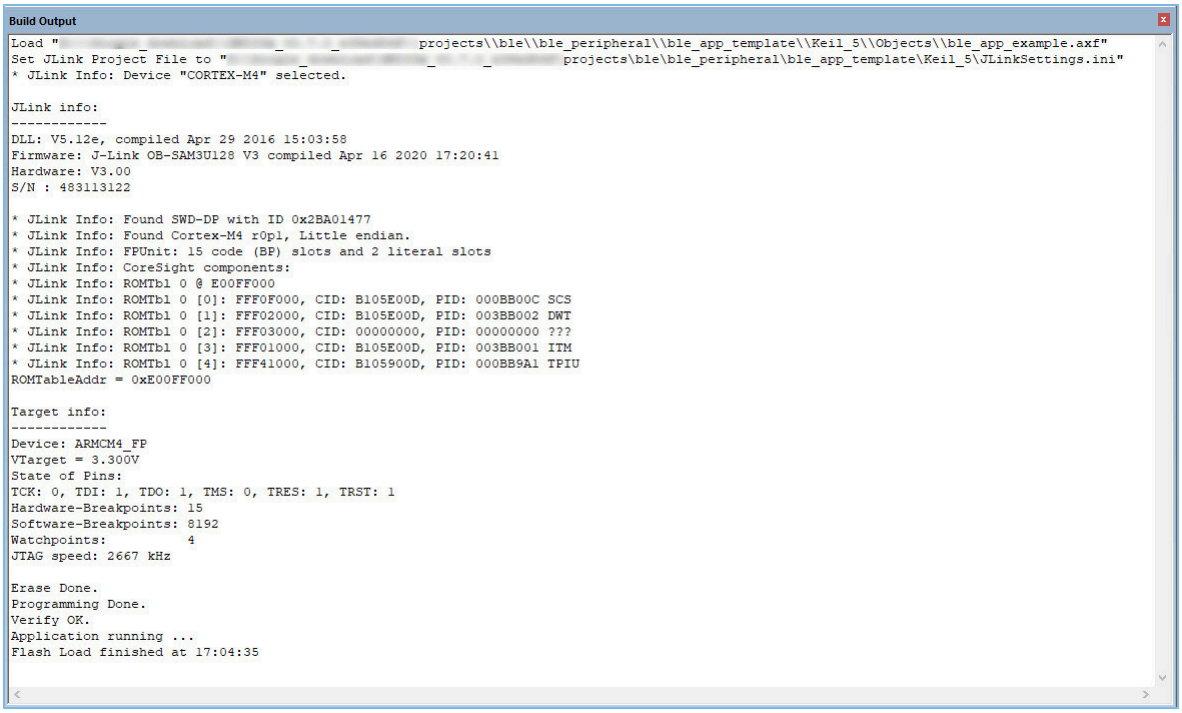

图 4-12 下载结果

说明:

下载过程中，若界面提示“No Cortex-M SW Device Found”，则表示芯片当前可能处于睡眠状态（即开启睡眠模式的工程正在运行），无法直接下载.hex文件到Flash中。开发者需先按下GR5405 SK板的“RESET”键，间隔1秒左右，再点击“Download”按钮，重新下载文件。

4.6 调试

Keil提供了调试器，支持代码在线调试。该调试器支持设置6个硬件断点和多个软件断点。开发者还可以使用多种方式设置调试命令。

4.6.1 配置调试器

启动调试之前，需要配置调试器。点击Keil 工具栏中的“Options for Target”按钮，打开“Options for Target ‘GRxx_Soc’ ”对话框，选择“Debug”标签页。窗口左侧为软件仿真调试配置，右侧为硬件在线调试配置。

Bluetooth LE Examples工程使用硬件在线调试，相关的调试器默认配置如下：

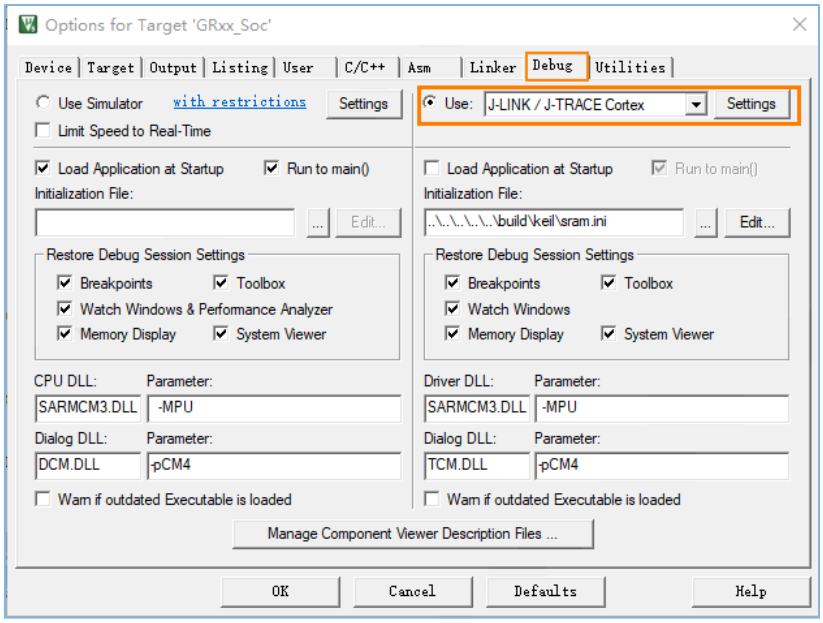

图 4-13 配置调试器

默认使用的Initialization File（*sram.ini*）位于SDK_Folder\build\keil目录。开发者可以直接使用该文件，也可以将该文件复制到自己的工程目录下使用。

初始化文件*sram.ini*中包含一组调试命令，调试过程将执行这些命令。点击“Initialization File”栏右侧的“Edit...”按钮，可打开*sram.ini*文件。

*sram.ini*的代码示例如下：

```
/**
*************************************************************************
*GR55xx object loading script through debugger interface (e.g.Jlink, etc).
*The goal of this script is to load the Keils's object file to the GR55xx RAM
*assuring that the GR55xx has been previously cleaned up.
*************************************************************************
*/

//Debugger reset(check Keil debugger settings)
//Preselected reset type (found in Options->Debug->Settings)is Normal(0);
//-Normal:Reset core & peripherals via SYSRESETREQ & VECTRESET bit
RESET

//Load current object file
LOAD %L

//Load stack pointer
SP = _RDWORD(0x00000000)

//Load program counter
$ = _RDWORD(0x00000004)

//Write 0 to vector table register, remap vector
```

```
_WDWORD(0xE000ED08, 0x00000000)

//_WDWORD(0xE000E180, 0xFFFFFFFF)

//Write run address to 0xA000C578 register,For the debug mode;
//boot code will check the value of 0xA000C578 firstly,if the value of 0xA000C578 is
 valid,gr551x will jump to run

//_WDWORD(0xA000C578, 0x00810000)
```

说明:

Keil支持按照以下顺序执行开发者设置的调试器命令：

1. 当“Options for Target ‘GRxx_Soc’ > Debug > Load Application at Startup”被使能，调试器会首先载入“Options for Target ‘GRxx_Soc’ > Output > Name of Executable”中的文件。
2. 执行“Options for Target ‘GRxx_Soc’ > Debug > Initialization File”所指定文件中的命令。
3. 当“Options for Target ‘GRxx_Soc’ > Debug > Restore Debug Session Settings”包含的选项被选中，恢复相应的Breakpoints、Watch Windows、Memory Display等。
4. 当“Options for Target ‘GRxx_Soc’ > Debug > Run to main()”被选中或者命令`g,main`被发现位于Initialization File中，调试器就开始自动执行CPU指令，直至遇到main()才会停下来。

4.6.2 启动调试

完成调试器配置后，点击Keil 工具栏中的“Start/Stop Debug Session”按钮  ，即可开始调试。

说明:

“Connect”需设置为“Normal”，“Reset Options”设置为“Core”（如下图所示），以保证启动“Debug Session”之后，点击Keil工具栏中的“Reset”按钮，程序仍能正常运行。

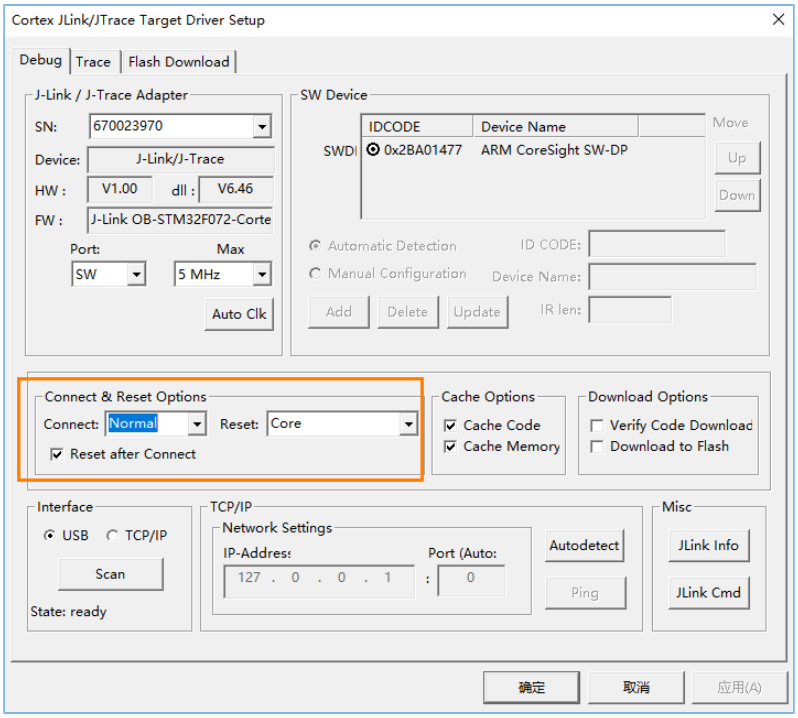

图 4-14 “Connect & Reset Options”配置

4.6.3 输出调试Log

GR5405 SDK提供APP LOG模块，可支持硬件端口输出Application调试Log，并支持开发者自定义输出方式：UART、J-Link RTT或ARM ITM（Instrumentation Trace Macrocell）。

若使用APP Log模块，则需在*custom_config.h*中打开宏APP_LOG_ENABLE，并根据所需输出方式配置宏APP_LOG_PORT。

4.6.3.1 模块初始化

完成配置后，开发者还需在外设初始化阶段，调用`app_log_init()`完成APP LOG模块的初始化，包括配置Log参数、注册Log输出接口和Flush接口。

APP LOG模块支持使用标准C库函数`printf()`和APP LOG API输出调试Log。若使用APP LOG API，则可通过设置Log级别、格式、过滤方式等参数优化Log输出；若使用`printf()`，则可将Log参数设置为NULL。

根据开发者所设置的输出方式，调用相应模块的初始化函数（具体可参考`SDK_Folder\platform\boards\board_SK.c`），并注册相应的Log输出接口和Flush接口，可参考函数bsp_log_init()。

以UART输出方式为例，bsp_log_init()实现如下：

```
void bsp_log_init(void)
{
#if APP_LOG_ENABLE

#if (APP_LOG_PORT == 0)
    bsp_uart_init();
#elif (APP_LOG_PORT == 1)
    SEGGER_RTT_ConfigUpBuffer(0, NULL, NULL, 0, SEGGER_RTT_MODE_NO_BLOCK_TRIM);
```

```
#endif

#if (APP_LOG_PORT <= 2)
    app_log_init_t   log_init;

    log_init.filter.level                  = APP_LOG_LVL_DEBUG;
    log_init.fmt_set[APP_LOG_LVL_ERROR]    = APP_LOG_FMT_ALL & (~APP_LOG_FMT_TAG);
    log_init.fmt_set[APP_LOG_LVL_WARNING] = APP_LOG_FMT_LVL;
#ifdef APP_LOG_NO_PFX
    log_init.fmt_set[APP_LOG_LVL_INFO]     = APP_LOG_FMT_NULL;
    log_init.fmt_set[APP_LOG_LVL_DEBUG]    = APP_LOG_FMT_NULL;
#else
    log_init.fmt_set[APP_LOG_LVL_INFO]     = APP_LOG_FMT_LVL;
    log_init.fmt_set[APP_LOG_LVL_DEBUG]    = APP_LOG_FMT_LVL;
#endif

#if (APP_LOG_PORT == 0)
    app_log_init(&log_init, bsp_uart_send, bsp_uart_flush);
    app_log_assert_flush_init(bsp_uart_assert_flush);
#elif (APP_LOG_PORT == 1)
    app_log_init(&log_init, bsp_segger_rtt_send, NULL);
#elif (APP_LOG_PORT == 2)
    app_log_init(&log_init, bsp_itm_send, NULL);
#endif /* APP_LOG_PORT */

#endif /* APP_LOG_PORT <= 2 */

#endif /* APP_LOG_ENABLE == 1 */

    app_assert_init();
}
```

说明:

- app_log_init()接口的输入参数包括Log初始化参数、Log输出接口和Flush接口。
- app_log_assert_flush_init()接口的输入参数为专用于APP ASSERT模块的Flush接口。其中，APP ASSERT模块的实现位于SDK_Folder\components\libraries\app_assert。

当使用UART输出调试Log时，已实现的Log输出接口和Flush接口分别为bsp_uart_send()和bsp_uart_flush()。

- bsp_uart_send()实现了app_uart同步（app_uart_transmit_sync接口）和app_uart异步（app_uart_transmit_async接口）两种方式的输出接口，可通过增加/删除宏APP_LOG_ASYNC进行控制。请根据实际需求，选择合适的Log输出方式。
- bsp_uart_flush()为uart_flush接口，用于app_uart异步方式下输出内存中缓存的Log数据。
- bsp_uart_assert_flush()为uart_assert_flush接口，用于发生ASSERT时输出内存中缓存的Log数据。

说明:

开发者可重写上述三个接口函数。

当使用J-Link RTT或ARM ITM输出调试Log时，已实现的Log输出接口分别为bsp_segger_rtt_send() 和bsp_itm_send()。在这两种模式下，没有实现Flush接口。

4.6.3.2 使用方法

APP LOG模块初始化完成后，开发者可以使用以下四个API输出调试Log：

- APP_LOG_ERROR()
- APP_LOG_WARNING()
- APP_LOG_INFO()
- APP_LOG_DEBUG()

如果使用中断输出模式，可调用app_log_flush()函数将缓存中的全部调试Log输出，从而保证芯片复位或系统睡眠之前，输出全部调试Log。

当选择使用J-Link RTT方式输出Log时，推荐在*SEGGER_RTT.c*中做如下修改：

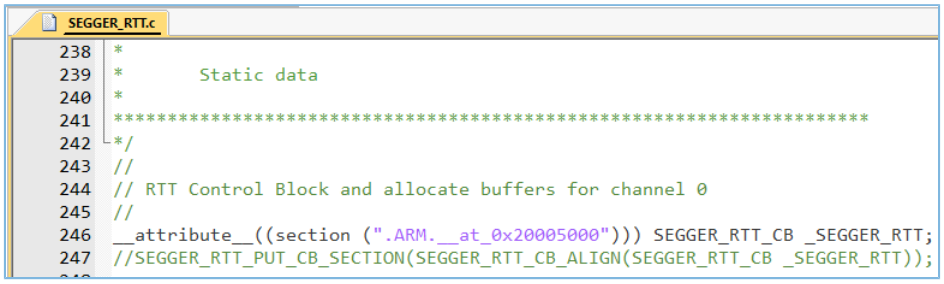

图 4-15 创建RTT Control Block并置于地址0x20005000处

若需对J-Link RTT Viewer进行配置，则可参考图 4-16中的参数配置。

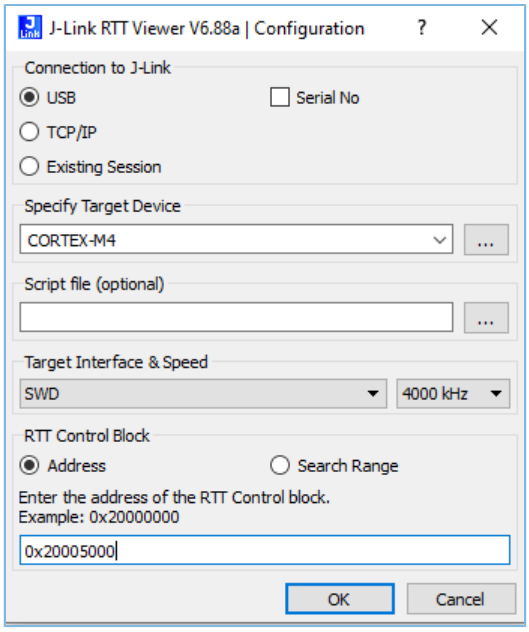

图 4-16 配置J-Link RTT Viewer

其中“RTT Control Block”的地址可通过“Address”方式指定，输入值可设置为编译工程生成的.map映射文件中“_SEGGER_RTT”结构体的地址，如下图所示。若按图 4-15推荐方式创建RTT Control Block并将其置于地址0x20005000处，则“Address”应设置为“0x20005000”。

```
ultra_wfi_or_wfe                0x200037ec   Data          0  rom_symbol.txt ABSOLUTE
sdk_gap_env                     0x200038ec   Data          0  rom_symbol.txt ABSOLUTE
_SEGGER_RTT                     0x20005000   Data        120  segger_rtt.o(.ARM.__at_0x20005000)
jlink_opt_info                  0x20006000   Data          0  rom_symbol.txt ABSOLUTE
SystemCoreClock                 0x2000b000   Data          4  system_gr55xx.o(.data)
__stdout                        0x2000b044   Data          4  app_log.o(.data)
```

图 4-17 获取RTT Control Block地址

4.6.4 使用GRToolbox调试

GR5405 SDK提供GRToolbox App （Android版），可用于GR5405 Bluetooth LE应用调试。该App主要提供以下功能：

- 通用的Bluetooth LE扫描和连接，对Characteristics的读写。
- 标准Profile的Demo展示，包括Heart Rate等。
- Goodix自定义应用程序。

🔔 提示:

GRToolbox安装文件可从汇顶官网获取，或从应用市场下载。

4.7 使用IAR下载、调试

GR5405 SDK的示例工程不仅包含Keil工程，还提供了IAR工程，可方便用户使用IAR工具快速编译示例工程，生成应用固件。例如，ble_app_template示例的IAR工程位于`SDK_Folder\projects\ble\ble_peripheral\ble_app_template\IAR`。编译*ble_app_template.eww*工程，可生成固件文件*ble_app_template.bin*。

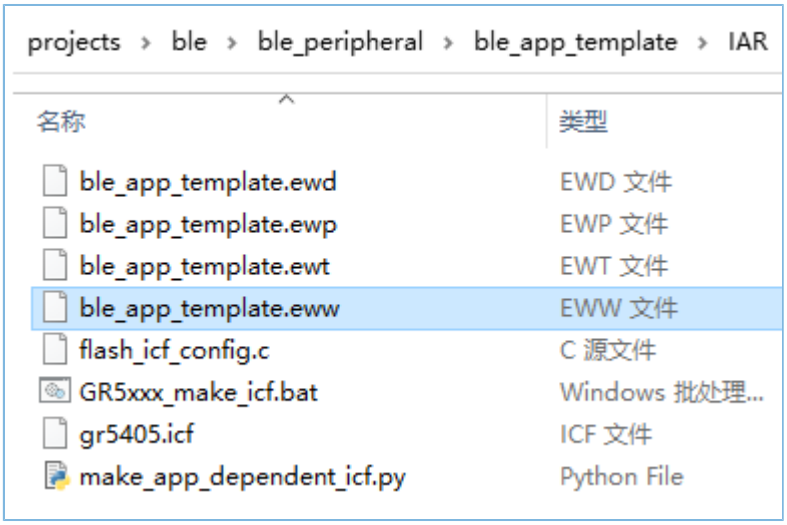

图 4-18 IAR工程目录

使用IAR下载和调试固件的操作步骤如下：

1. 在IAR IDE的菜单栏中，选择“Project > Options”，打开配置窗口；在左侧的“Category”列表中选中“Debugger”，然后在“Setup”页面的“Driver”栏中选择“J-Link/J-Trace”，并勾选“Use macro files(s)”，输入“$PROJ_DIR$\..\..\..\..\..\build\iar\GR5xxx.mac”。

2. 将`SDK_Folder/build/iar`路径下的下载算法文件*GR5xxx_IAR_16M.board*、*GR5xxx_IAR_16M.flash*以及*GR5xxx_IAR_flashloader_16M.out*拷贝至`IAR_Install/arm/config/flashloader/Goodix`（其中，IAR_Install为IAR软件安装目录，Goodix为新建文件夹）。

3. 在IAR IDE中，选择“Project > Options > Debugger > Download”，将*Override default .board file*设置为*GR5xxx_IAR_16M.board*文件（位于`IAR_Install/arm/config/flashloader/Goodix`）。

4. 在IAR IDE中，选择“Project > Options > Debugger > J-Link/J-Trace > Setup”，在“Reset”栏中选择“Core”。

5. 在IAR IDE的菜单栏中，选择“Project > Download > Download Active Application”，在弹出的对话框中点击“OK”按钮，即可开始下载固件，并进入调试模式。

5 术语与缩略语

表 5-1 术语与缩略语

名称	描述
API	Application Programming Interface，应用程序编程接口
ATT	Attribute Protocol，属性协议层
Bluetooth LE	Bluetooth Low Energy，低功耗蓝牙
DFU	Device Firmware Update，设备固件更新
DTM	Direct Test Mode，直接测试模式
DUT	Device Under Test，待测设备
GAP	Generic Access Profile，通用访问规范
GATT	Generic Attribute Profile，通用属性规范
GFSK	Gaussian Frequency Shift Keying，高斯频移键控
HAL	Hardware Abstract Layer，硬件抽象层
HCI	Host Controller Interface，主机-控制器接口
HPA	High Power Amplifier，高功率放大器
L2CAP	Logical Link Control and Adaptation Protocol，逻辑链路控制与适配协议
LL	Link Layer，链路层
NVDS	Non-volatile Data Storage，非易失性数据存储
PMU	Power Management Unit，电源管理单元
PHY	Physical Layer，物理层
RF	Radio Frequency，射频
SCA	System Configuration Area，系统配置区
SDK	Software Development Kit，软件开发工具包
SM	Security Manager，安全管理器
SoC	System-on-Chip，系统级芯片
SPA	Small Power Amplifier，小功率放大器
UART	Universal Asynchronous Receiver/Transmitter，异步收发传输器
XIP	Execute in Place，片上执行